Allocation methods, Directory implementation file system

What is File Allocation in OS?

When a hard drive is formatted, a system has numerous tiny spaces to store files called blocks. The OS stores data in memory blocks using various file allocation methods, which enables the hard disk to be used efficiently and the file to be accessed.

Types of File Allocation Methods in Operating System.

1. Contiguous File allocation

$$\left.\begin{array}{l}\text{2. Linked File Allocation}\\ \text{3. Indexed File Allocation}\end{array}\right\}\quad \text{Non Contiguous}$$

4. File Allocation Table (FAT)
5. Inode

Contiguous File Allocation

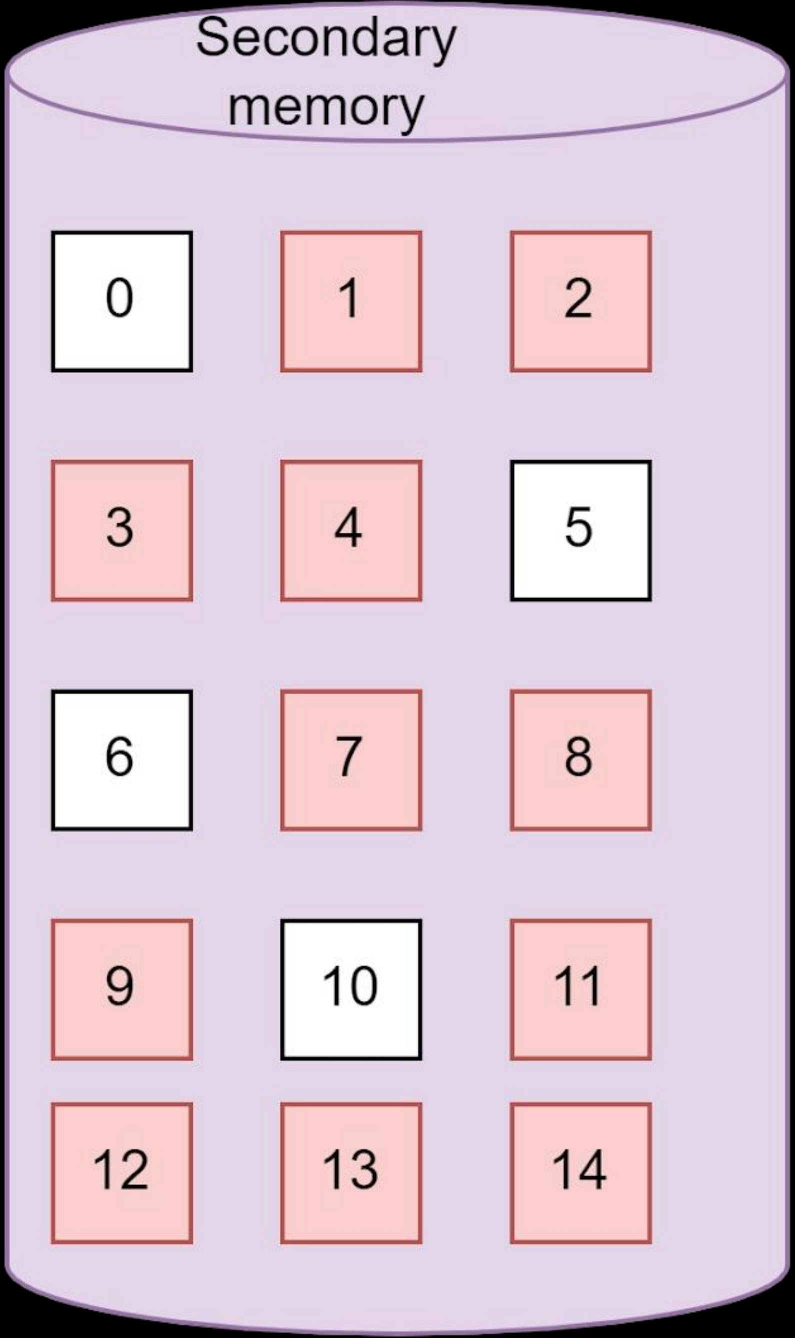
In the contiguous allocation methods, the files are allocated the disk blocks in a contiguous manner, meaning that if a file's starting disk block address is x, then it will be allocated the blocks with address x+1, x+2, x+3,……., provided the blocks are not already occupied.

Advantages

Easy to Implement
Excellent Read Performance

Disadvantages

Disk will become fragmented
Difficult to grow file



File	start	length
os.pdf	1	4
dbms.doc	7	3
dsa.mp4	11	4

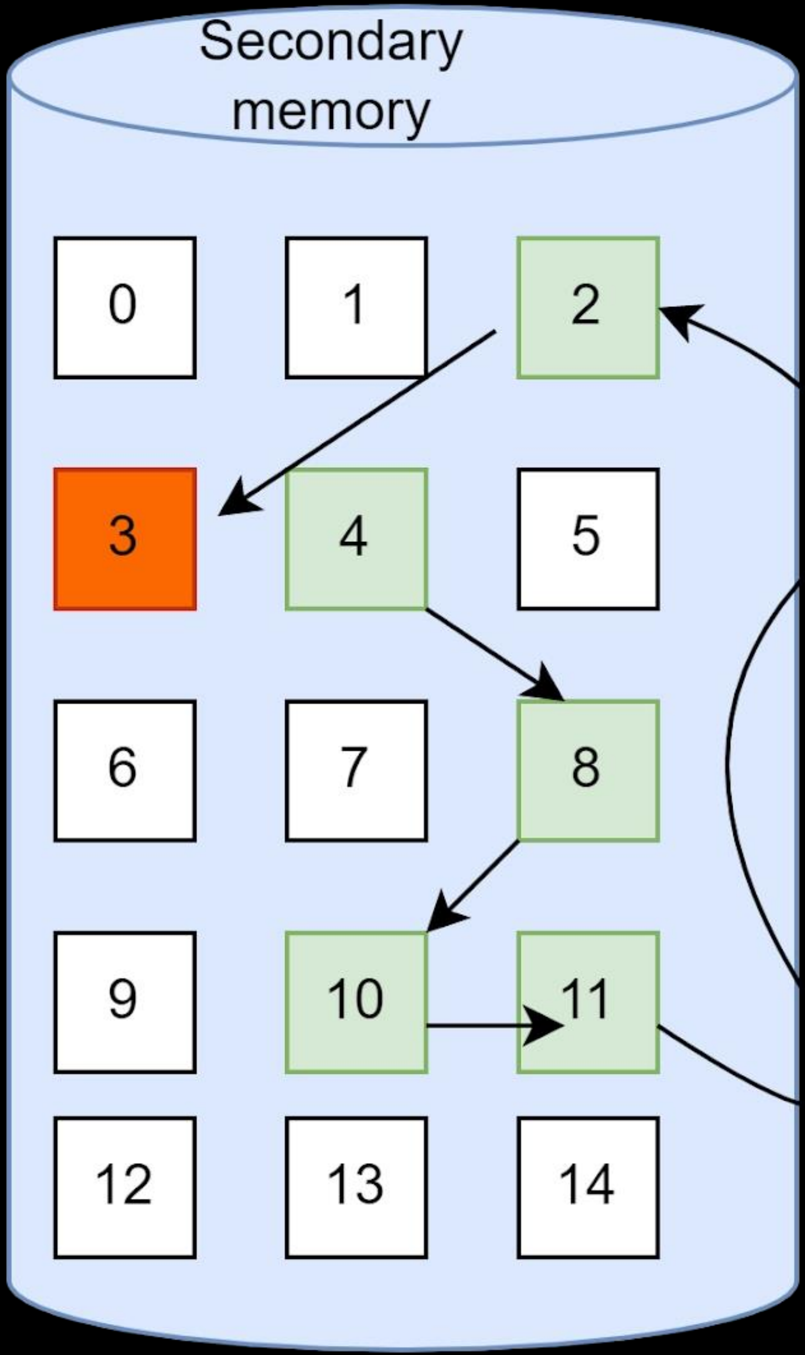
Linked List allocation

The linked allocation works just like the linked list. The problem with contiguous allocation was that memory remained unutilized due to external fragmentation.

The solution to the problem was to allocate the disk block in the form of a linked list where every block was a node.

The blocks are allocated in such a way that every block contains a pointer to the next block that is allocated to the file.

The above image shows how the linked allocation works. The file "os.pdf" has to be allocated some blocks.



File	start	end
os.pdf	4	3

The first block allocated is 4. Block 4 will have a pointer to the next block 8, block 8 will have a pointer to block 10, block 10 will have a pointer to block 11, block 11 will have a pointer to block 2, and finally, block 2 will point to 3.

In this manner, a total of six blocks are allocated to the file in a non-contiguous manner. The ending block (block 3) will not point to any other block.

Advantages

no external fragmentation

File size can be increased

Disadvantages

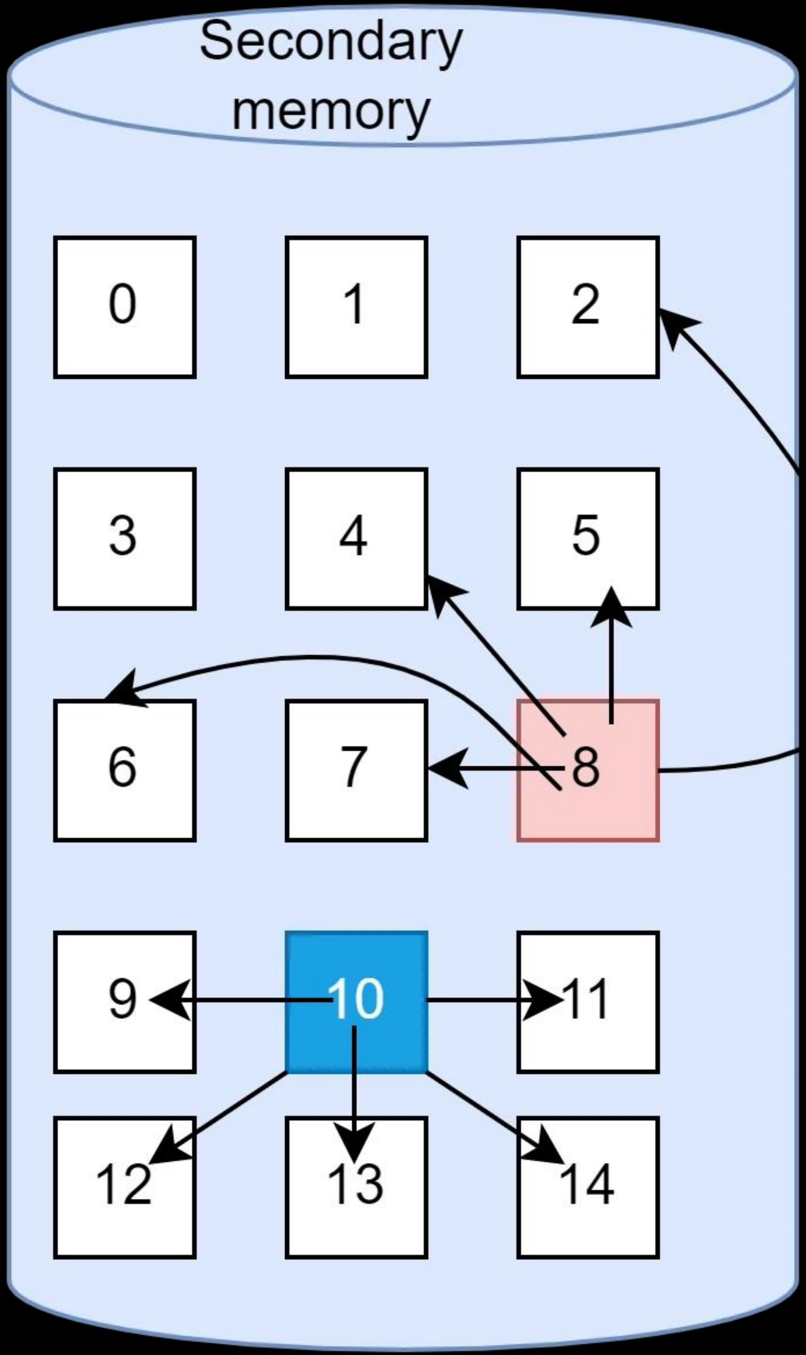
Random access is not allowed
slow because of more seek time
Every block needs to contain extra information about the pointer to the next block.

Indexed File Allocation

There are index blocks that contain pointers to the blocks occupied by the file.

Thus every file has its own index block, which is a disk block, but instead of storing the file itself, it contains the pointers to other blocks that store the file.

This helps in randomly accessing the files and also avoiding external fragmentation.



File name	Index block
os.pdf	8
dbms.doc	10

index block 8
5
4
2
6
7

From the above image, we can see that block number 8 does not store the file but contains the pointers to various other blocks, which store the file.

The directory table contains only the file name and the index block for the respective files.

Below is the pictorial representation of index block 8, which contains the pointers that determine the address of the blocks that store the "os.pdf" file.

Advantages

Support direct access

No External Fragmentation

Disadvantages

Pointer overhead

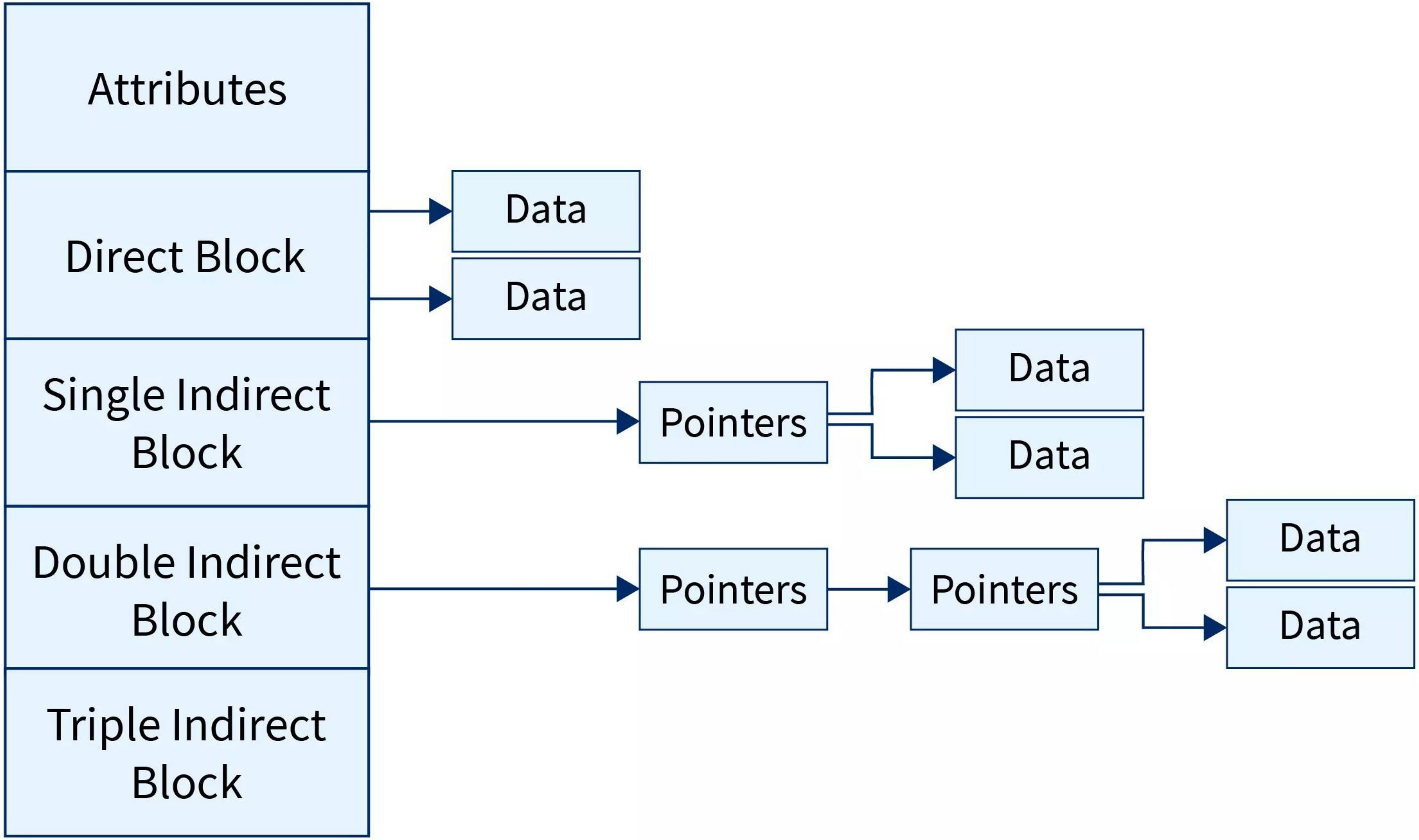
files that are very large, single index block may not be able to hold all the pointers.

Inode (Information-node) OR Unix Inode Structure

In Operating systems based on UNIX, every file is indexed using **Inode(information-node)**. The inode is the special disk block that is created with the creation of the file system.

This special block is used to store all the information related to the file like name, authority, size, etc along with the pointers to the other blocks where the file is stored.

The first few pointers in Inode point to **direct blocks**, i.e they contain the addresses of the disk blocks containing the file data. A few pointers in inode point to the **indirect blocks**. There are three types of indirect blocks: *single indirect*, *double indirect*, and *triple indirect*.


Attributes
Direct Block
Single Indirect Block
Double Indirect Block
Triple Indirect Block
Data
Data
Pointers
Data
Data
Pointers
Pointers
Data
Data

A **single indirect block** contains nothing but the address of the block containing the file data, not the file itself.

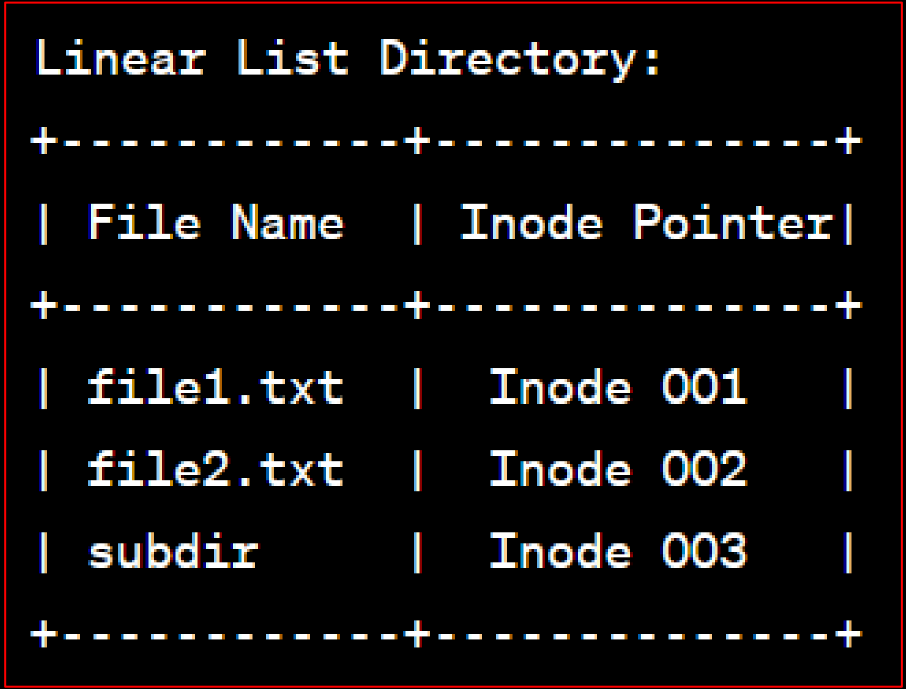
Double indirect block points to the pointers which again point to the pointers which point to the data blocks. Further, it goes in a similar way for **triple indirect block**.

Directory Implementation in Operating System

In operating systems, a directory is a file system component that organizes and provides a hierarchical structure for files and subdirectories. The implementation of directories involves data structures and algorithms to efficiently organize and manage information about files and their locations on storage devices. There are different approaches to directory implementation, and two common methods are the linear list and the hash table.

1. Linear List:

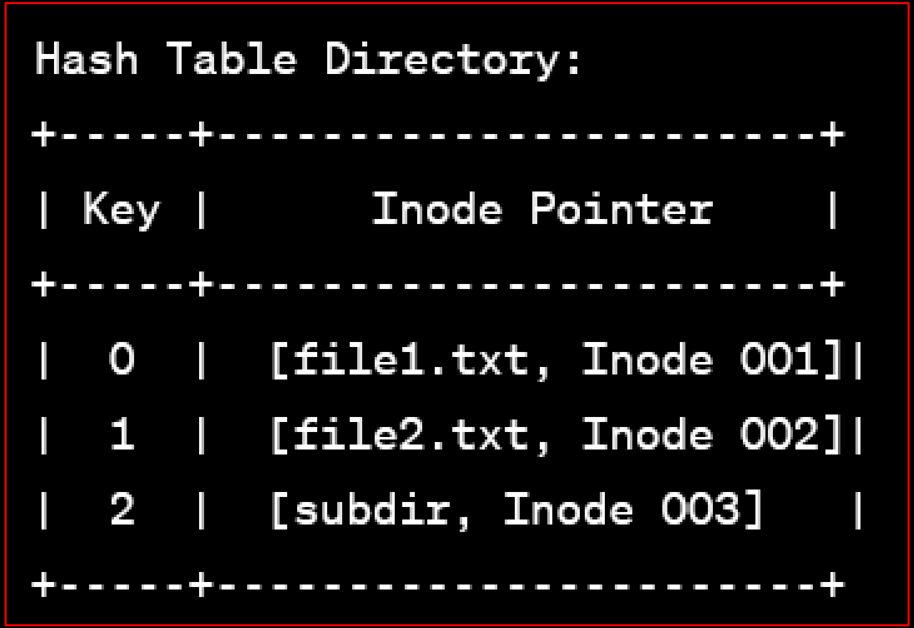
- In a linear list directory structure, entries for files and subdirectories are arranged in a linear list or array.
- Each entry typically contains the file or directory name and a pointer to its corresponding inode or file control block.
- The entries are stored sequentially, allowing for easy traversal.
- This method is straightforward but may lead to slower search times as the size of the directory grows.



```
Linear List Directory:
+------------+--------------+
| File Name  | Inode Pointer|
+------------+--------------+
| file1.txt  |  Inode 001   |
| file2.txt  |  Inode 002   |
| subdir     |  Inode 003   |
+------------+--------------+
```

2. Hash Table:

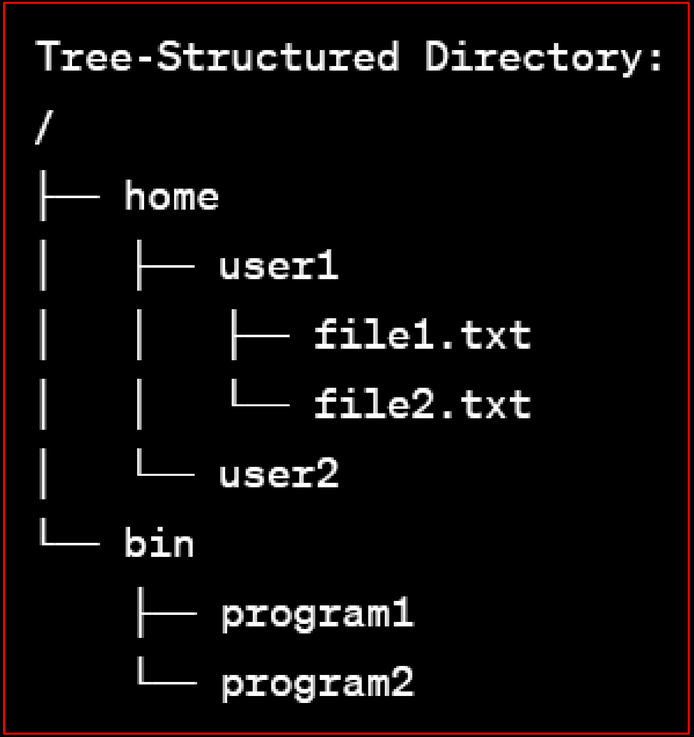
- A hash table directory uses a hash function to map file or directory names to specific locations in the table.
- This method aims to provide faster search times by reducing the search space.
- Collisions (when two names hash to the same location) can be handled using techniques like chaining or open addressing.
- The hash table is often implemented as an array of linked lists or buckets.



```
Hash Table Directory:
+-----+------------------------+
| Key |      Inode Pointer     |
+-----+------------------------+
|  0  |  [file1.txt, Inode 001]|
|  1  |  [file2.txt, Inode 002]|
|  2  |  [subdir, Inode 003]   |
+-----+------------------------+
```

3. Tree Structure:

- Directories are often implemented as tree structures, where each node represents a directory or file.
- The root of the tree is the top-level directory, and branches represent subdirectories.
- Leaves of the tree represent files.
- This structure allows for efficient organization and retrieval of information.



```
Tree-Structured Directory:
/
├── home
│   ├── user1
│   │   ├── file1.txt
│   │   └── file2.txt
│   └── user2
└── bin
    ├── program1
    └── program2
```