

## 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.

All the three methods have their own advantages and disadvantages as discussed below:

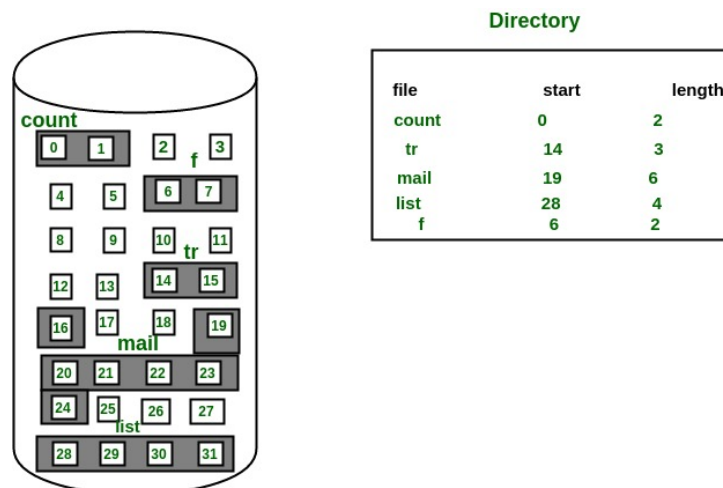
### 1. Contiguous Allocation

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:  $b, b+1, b+2, \dots, b+n-1$ . This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

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

The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.



### Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the  $k$ th 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:

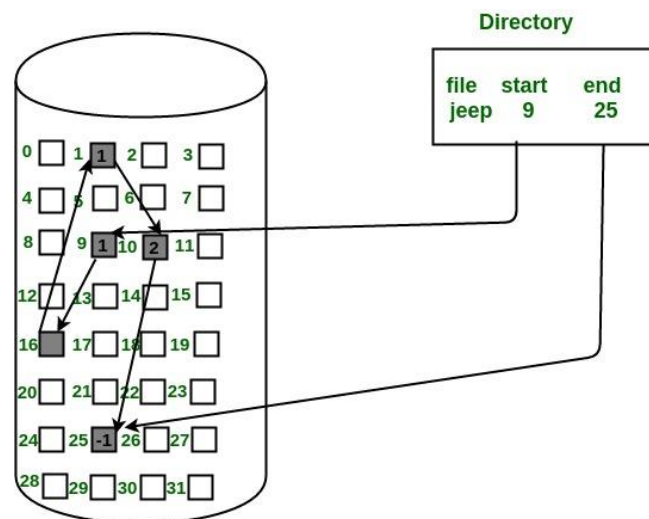
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

## 2. 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.

*The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.*



#### Advantages:

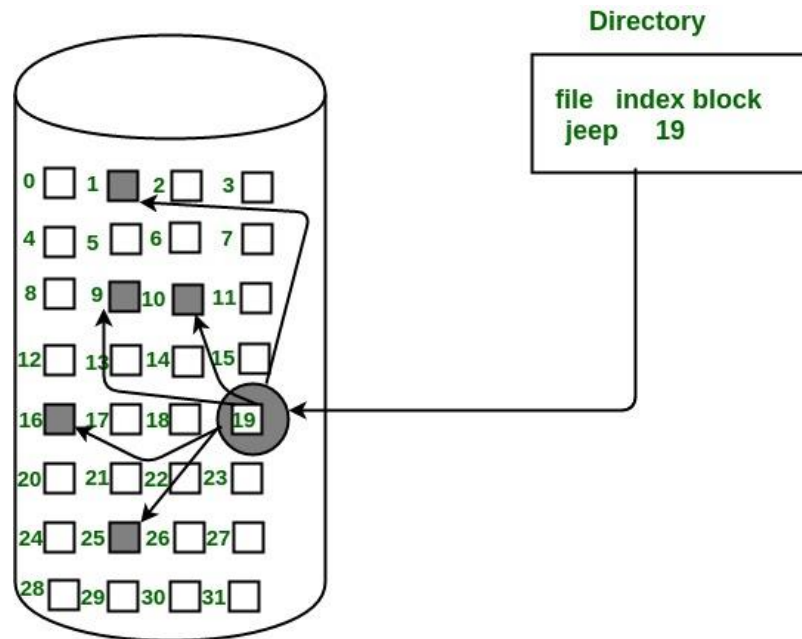
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

#### 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.

### 3. Indexed Allocation

In this scheme, 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 *i*th entry in the index block contains the disk address of the *i*th 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.

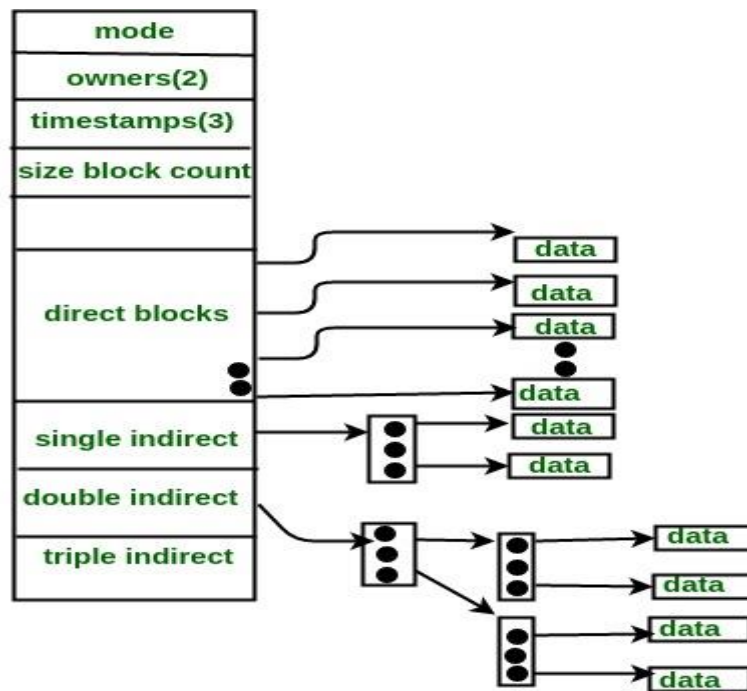
#### Disadvantages:

- 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.

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 a pointer or the address 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 in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:** In this scheme, a special block called the **Inode (information Node)** contains all the information 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 *as shown in the image below*. The first few of these pointers in Inode point 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 point 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. Similarly, **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.



Indexed allocation is a file allocation method that can minimize the amount of space required in a directory structure. It's a combination of linked and contiguous allocation that's more efficient in terms of space and time.

In indexed allocation, files are stored in non-contiguous blocks. The operating system creates an index block that contains a list of pointers to each block in the file.

The three main methods of allocating disk space are: Contiguous allocation, Linked allocation, Indexed allocation.

## Free space management in Operating System

Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices. The operating system uses various techniques to manage free space and optimize the use of storage devices. Here are some of the commonly used free space management techniques:

1. **Linked Allocation:** In this technique, each file is represented by a linked list of disk blocks. When a file is created, the operating system finds enough free space on the disk and links the blocks of the file to form a chain. This method is simple to implement but can lead to fragmentation and wastage of space.
2. **Contiguous Allocation:** In this technique, each file is stored as a contiguous block of disk space. When a file is created, the operating system finds a contiguous block of free space and assigns it to the file. This method is efficient as it minimizes fragmentation but suffers from the problem of external fragmentation.
3. **Indexed Allocation:** In this technique, a separate index block is used to store the addresses of all the disk blocks that make up a file. When a file is created, the operating system creates an index block and stores the addresses of all the blocks in the file. This method is efficient in terms of storage space and minimizes fragmentation.
4. **File Allocation Table (FAT):** In this technique, the operating system uses a file allocation table to keep track of the location of each file on the disk. When a file is created, the operating system updates the file allocation table with the address of the disk blocks that make up the file. This method is widely used in Microsoft Windows operating systems.
5. **Volume Shadow Copy:** This is a technology used in Microsoft Windows operating systems to create backup copies of files or entire volumes. When a file is modified, the operating system creates a shadow copy of the file and stores it in a separate location. This method is useful for data recovery and protection against accidental file deletion.

Overall, free space management is a crucial function of operating systems, as it ensures that storage devices are utilized efficiently and effectively.

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, 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 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. The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

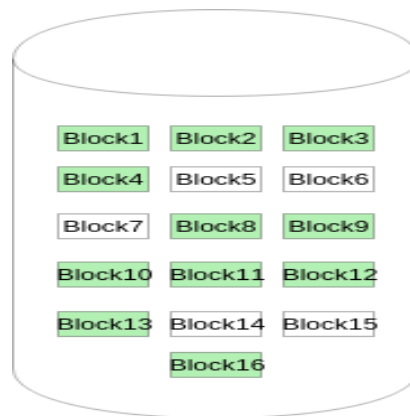


Figure - 1

### Advantages –

- Simple to understand.
  - Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.
2. **Linked List** – In this approach, 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.

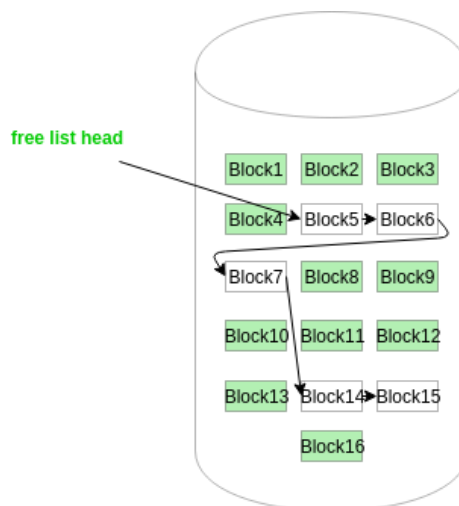


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list. A drawback of this method is the I/O required for free space list traversal.

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 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:
  - Address of first free disk block
  - A number  $n$

## Some advantages and disadvantages of free space management techniques

### Advantages:

1. Efficient use of storage space: Free space management techniques help to optimize the use of storage space on the hard disk or other secondary storage devices.
2. Easy to implement: Some techniques, such as linked allocation, are simple to implement and require less overhead in terms of processing and memory resources.
3. Faster access to files: Techniques such as contiguous allocation can help to reduce disk fragmentation and improve access time to files.

### Disadvantages:

1. Fragmentation: Techniques such as linked allocation can lead to fragmentation of disk space, which can decrease the efficiency of storage devices.
2. Overhead: Some techniques, such as indexed allocation, require additional overhead in terms of memory and processing resources to maintain index blocks.
3. Limited scalability: Some techniques, such as FAT, have limited scalability in terms of the number of files that can be stored on the disk.
4. Risk of data loss: In some cases, such as with contiguous allocation, if a file becomes corrupted or damaged, it may be difficult to recover the data.
5. Overall, the choice of free space management technique depends on the specific requirements of the operating system and the storage devices being used. While some techniques may offer advantages in terms of efficiency and speed, they may also have limitations and drawbacks that need to be considered.