1. Explain different directory implementation methods.

Directory implementation involves a selection of efficient and reliable directory allocation and management algorithms that enhance the performance of the system. [There are mainly two algorithms for implementing directory, by using linear list and hash table](https://www.geeksforgeeks.org/directory-implementation-in-operating-system/):

**Linear list**

The linear list method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file, we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory or at the beginning of the directory. To delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it. To reuse the directory entry we can mark that entry as unused or we can append it to the list of free directories.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Simple to implement | Slow access to files |
| Easy to delete files | Difficult to insert files |
| No extra space required | No grouping of files |

**Hash table**

The hash table method of implementing a directory is to use a hash table along with the linear list. Here the linear list stores the directory entries, but a hash data structure is used in combination with the linear list. In the hash table for each pair in the directory key-value pair is generated. The hash function on the file name determines the key and this key points to the corresponding file stored in the directory. This method efficiently decreases the directory search time as the entire list will not be searched on every operation. Using the keys the hash table entries are checked and when the file is found it is fetched.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Fast access to files | Complex to implement |
| Easy to insert and delete files | Requires extra space for storing keys or addresses |
| Allows grouping of files | Prone to data corruption or fragmentation |

Here is a neat sketch of the two directory implementation methods:

Linear list:

+-----------------+

| Directory |

+-----------------+

| File1 -> +-----> Data blocks

| File2 -> +-----> Data blocks

| File3 -> +-----> Data blocks

+-----------------+

Hash table:

+-----------------+ +-----------------+

| Hash table | | Directory |

+-----------------+ +-----------------+

| Key1 -> +---->| File1 -> +-----> Data blocks

| Key2 -> +---->| File2 -> +-----> Data blocks

| Key3 -> +---->| File3 -> +-----> Data blocks

+-----------------+ +-----------------+

2.  Describe in detail about free space management with neat.

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 devices1](https://www.geeksforgeeks.org/free-space-management-in-operating-system/)[2](https://www.scaler.com/topics/free-space-management-in-os/)[3](https://www.codingninjas.com/codestudio/library/free-space-management).

The operating system maintains a free space list to keep track of the free disk space. The free space list consists of all free disk blocks that are not allocated to any file or directory. The operating system uses the free space list to allocate space to files when they are created and to deallocate space when they are deleted.

There are various methods using which a free space list can be implemented. Some of the commonly used methods are:

**Bitmap or bit vector**

A bitmap or bit vector is a series or collection of bits in which each bit represents 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. This method is simple and easy to implement but requires extra space for storing the bitmap.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Simple and easy to implement | Requires extra space for storing bitmap |
| Fast and easy to find free blocks | Wastes space for large disks |
| Suitable for fixed-size allocation | Difficult to group contiguous blocks |

**Linked list**

A linked list is a data structure that stores the addresses of free disk blocks in a linked manner. Each block contains a pointer to the next free block. The first block of the list is pointed by a head pointer. This method does not require extra space for storing the free space list but requires extra space for storing pointers in each block.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Does not require extra space for storing free space list | Requires extra space for storing pointers in each block |
| Easy to insert and delete free blocks | Slow and sequential access to free blocks |
| Suitable for variable-size allocation | Prone to fragmentation and corruption |

**Grouping**

Grouping is a method that combines the advantages of bitmap and linked list methods. In this method, the disk is divided into groups of blocks. Each group has a bitmap that indicates the status of each block in the group. The first block of each group contains a pointer to the next group that has free blocks. This method reduces the size of bitmap and improves the access time.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Reduces the size of bitmap | Requires extra space for storing pointers in each group |
| Improves the access time | Difficult to group contiguous blocks |
| Suitable for variable-size allocation | Prone to fragmentation and corruption |

Here is a neat sketch of the three methods of free space management:

Bitmap:

+-----------------+

| Bitmap |

+-----------------+

| 0000111000000110|

+-----------------+

/ \

/ \

/ \

+---+ +---+

| A +---->+ B +----> Disk blocks

+---+ +---+

Linked list:

+-----------------+

| Head |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| A +---->+ B +----> Free disk blocks

+---+ +---+

Grouping:

+-----------------+

| Head |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| A +---->+ B +----> Groups of disk blocks

+---+ +---+

/ \ / \

+-+-+ +-+-+

|BMP| |BMP|

+-+-+ +-+-+

3.Explain in detail about tree structured and acyclic graph directories.

[Tree-structured and acyclic-graph directories are two logical structures of directories that allow users to create their own subdirectories and to organize their files accordingly1](http://boron.physics.metu.edu.tr/ozdogan/OperatingSystems/week11/node18.html)[2](https://www.javatpoint.com/os-acyclic-graph-directories)[3](https://www.electronicsmind.com/directory-structure-in-operating-system/) .

**Tree-structured directory**

The tree-structured directory allows users to create their own subdirectories and to organize their files accordingly. A tree is a data structure that consists of nodes connected by edges. A tree has one root node and can have zero or more child nodes. Each child node can have zero or more child nodes, and so on. The tree-structured directory has one root directory, and every file in the system has a unique path name.

| **Advantages** | **Disadvantages** |
| --- | --- |
| No name collision | Path names can be long |
| Flexible grouping of files | Moving files and directories can be complex |
| Hierarchical organization of files | No sharing of files or directories |

**Acyclic-graph directory**

The acyclic-graph directory allows directories to share subdirectories and files. This is useful when two different users or groups want to access the same file without making copies of it. The acyclic-graph directory is similar to the tree-structured directory, except that it allows links or pointers to other directories or files. A link is essentially a pointer to another file or subdirectory.

| **Advantages** | **Disadvantages** |
| --- | --- |
| No duplication of files | Difficult to maintain consistency |
| Efficient use of disk space | Possible aliasing problems |
| Multiple access paths for files | Complex deletion of files or directories |

Here is a neat sketch of the two directory structures:

Tree-structured:

+-----------------+

| Root |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| A | | B |

+---+ +---+

/ \ / \

+---+ +---+

| C | | D |

+---+ +---+

Acyclic-graph:

+-----------------+

| Root |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| A +---->+ B +<----+

+---+ +---+ |

/ \ / \ |

+---+ +---+ v

| C +---->+ D +---->+ E +

+---+ +---+ +---+