DEPARTMENT OF AIML

23CST202- OPERATING SYSTEMS

II YEAR IV SEM AIML-B

UNIT 4-FILE SYSTEM

TOPIC –DIRECTORY IMPLEMENTATION

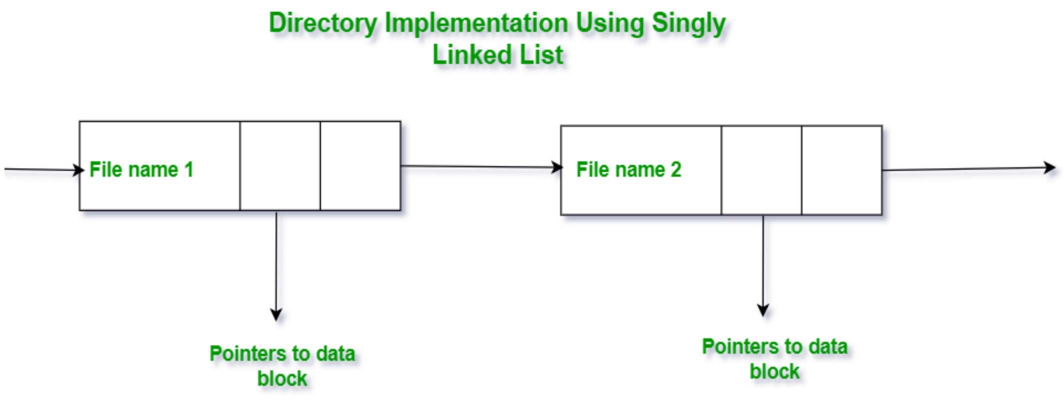
Directory Implementation in Operating System

Directory implementation in the operating system can be done using Singly Linked List and Hash table. The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms. There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.
The below are two ways of implementing the directory in the operating system :

- Directory Implementation using Singly Linked List
- Directory Implementation using Hash Table

Directory Implementation using Singly Linked List

The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. Here we implement a directory by using a linear list of filenames with pointers to the data blocks.



Directory Implementation Using Singly Linked List
Steps to Implement the Directory Using Singly Linked List
The steps are given below for the implementation of the directory:

- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.

- In order 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.
- To delete a file linked list is the best choice as it takes less time.

Advantages

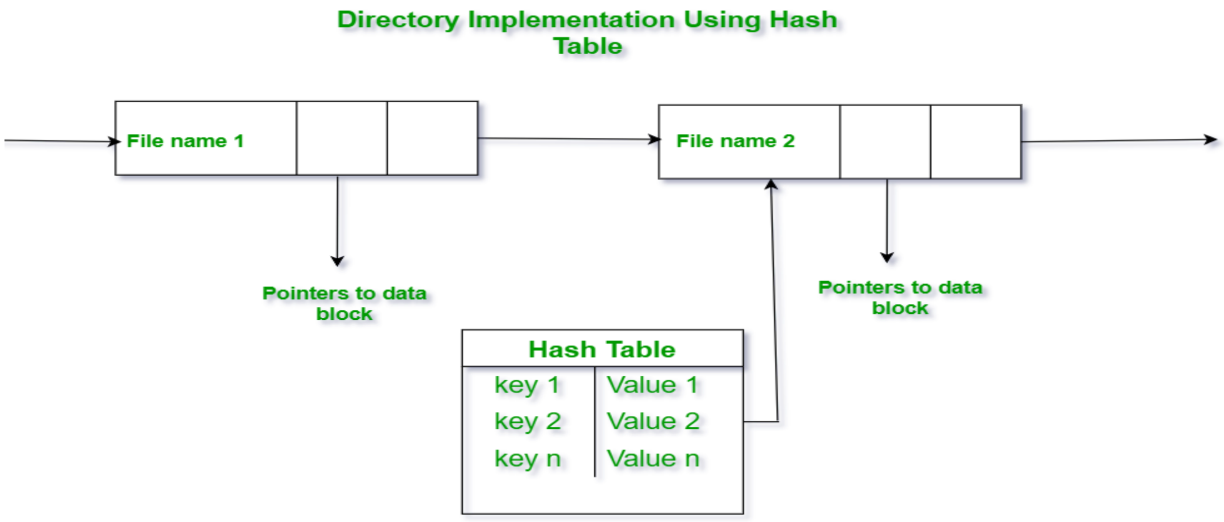
- Simple Implementation: Easy to implement with low memory overhead.
- Dynamic Structure: Grows or shrinks as needed without fixed size constraints.
- Efficient for Small Directories: Works well when the number of files is small and manageable.
- Low Complexity: No need for collision handling or resizing, making it simpler.

Disadvantages

- Lookup Time : File lookup requires a linear search, which can be time-consuming.
- Impact of Frequent Access : Directory information is accessed frequently, leading to slow access times with larger directories.
- Solution: Caching : Operating systems maintain a cache of recently accessed entries to enable quicker access without full traversal.

Directory Implementation using Hash Table

An alternative data structure that can be used for directory implementation is a hash table. It overcomes the major drawbacks of directory implementation using a linked list. In this method, we use a hash table along with the linked list. Here the linked list stores the directory entries, but a hash data structure is used in combination with the linked list.



Directory Implementation Using Hash Table

Steps to Implement the Directory Using Hash Table

The following steps are taken for the implementation of the directory using the hash table :

- Combine a hash table with a linked list to implement the directory structure.
- Generate a key-value pair for each file using a hash function on the file name.
- Insert the file into the linked list and store the key-pointer pair in the hash table.
- To search, compute the key using the file name and look it up in the hash table.
- Fetch the file directly using the pointer from the hash table, avoiding full list traversal.
- This hybrid method significantly reduces search time and improves efficiency.

Advantages

- Fast File Lookup: Provides average O(1) time complexity for quick search and retrieval.
- Efficient for Large Directories: Handles large directories with many files without significant performance loss.
- Scalable: Easily accommodates an increasing number of files without degrading access speed.
- Reduced Search Time: Eliminates the need for full traversal, making directory operations faster.

Disadvantage

- Fixed Size: Limited scalability due to a fixed size, affecting performance as data grows.
- Size Dependent Performance: Performance degrades as the table becomes full (high load factor).
- Collision Handling Complexity: Collisions add complexity and can slow down performance.
- Performance Trade off: Despite drawbacks, hash tables are faster than linked lists for lookups.

Comparison of Singly Linked List and a Hash Table Directory Implementation

Feature	Singly Linked List	Hash Table
Search Efficiency	Slow to find something, you need to check each item one by one O(n).	Fast , you can quickly find things because of how it's organized (O(1) on average).
Insertion	Fast and easy , you can add items at the beginning or end easily O(1).	Also fast O(1) , usually takes constant time, but can be slower if collisions happen.
Memory Usage	Low , it only stores the data and pointers.	Higher , uses more memory because it stores both data and additional space for handling collisions or empty slots.
Scalability	Not great for large data , as the list gets bigger, searching takes longer.	Good for large data , even as the data grows, it remains fast at finding things.
Implementation Complexity	Simple , easy to build and use.	More complex , you need to manage how data is hashed and handle collisions.
Collision Handling	Not needed , each element has its own place.	Needed , when two elements hash to the same spot, you need a way to handle it.

Feature	Singly Linked List	Hash Table
Adaptability	Easy to expand , just keep adding elements.	Not as flexible , resizing and rehashing can be costly if the table gets too full.
Best Use Case	Small lists with not too many elements.	Large lists or databases where speed is important.
Example Use	Small systems with limited memory (like embedded devices).	Large databases, file systems, or applications needing fast lookups.