Kernel I/O Subsystem in Operating System

- The kernel provides many services related to I/O.
- Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel
- The I/O subsystem is also responsible for protecting itself from the errant processes and malicious users.

I/O Scheduling

- To schedule a set of I/O requests means to determine a good order in which to execute them.
- Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, reduce the average waiting time, response time, turnaround time for I/O to complete.
- OS developers implement scheduling by maintaining a wait queue of the request for each device.
- When an application issues a blocking I/O system call, The request is placed in the queue for that device.

Buffering

- A *buffer* is a memory area that stores data being transferred between two devices or between a device and an application.
- Used to cope with a speed mismatch between producer and consumer of a data stream.

Caching

- A *cache* is a region of fast memory that holds a copy of data.
- Access to the cached copy is much easier than the original file.
- For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU's secondary and primary cache.

Spooling and Device Reservation

- A spool is a buffer that holds the output of a device, such as a printer.
- Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.
- The OS solves this problem by preventing all output continuing to the printer. The output of all applications is spooled in a separate disk file. When an application finishes printing then the spooling system queues the corresponding spool file for output to the printer.

Error Handling

I/O Protection

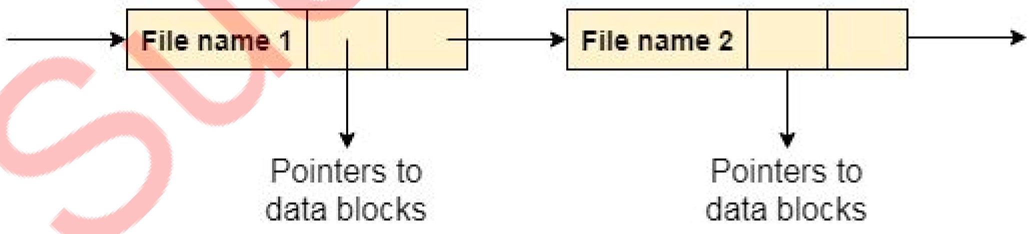
- To prevent illegal I/O access, we define all I/O instructions to be privileged instructions.

Directory Implementation

- There is the number of algorithms by using which, the directories can be implemented.
- The directory implementation algorithms are classified according to the data structure they are using.
- There are mainly two algorithms which are used these days.

1. Linear List

- In this algorithm, all the files in a directory are maintained as a singly linked list.
- Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.
- When a new file is created, then the entire list is checked whether the new file name is matching an existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end.
- Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
- The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

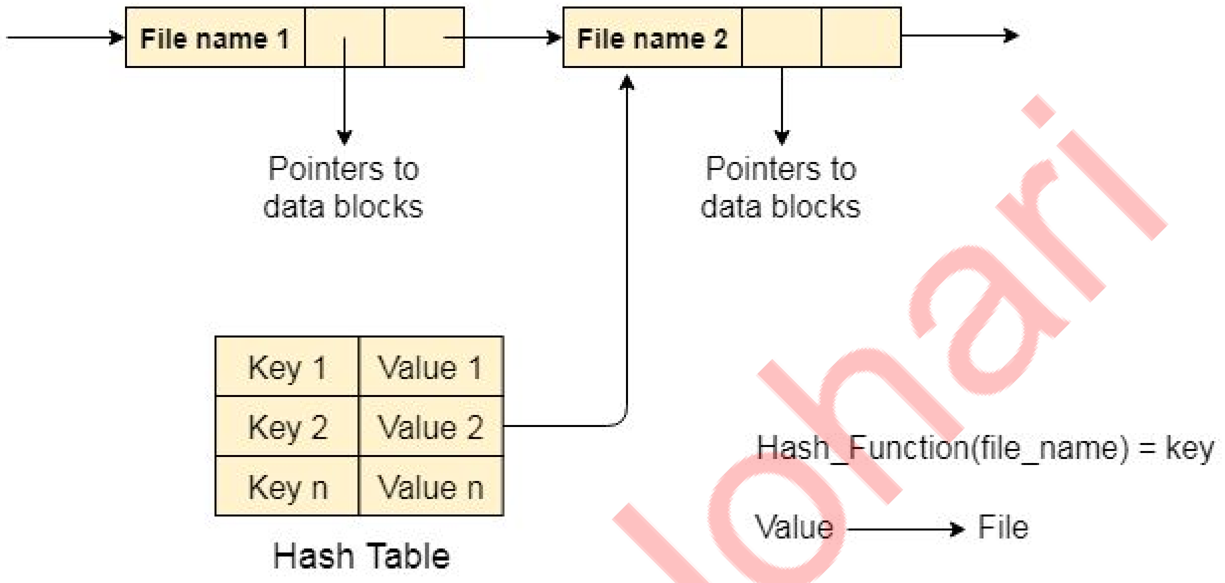


Linear List

2. Hash Table

- This approach suggests to use a hash table along with the linked lists.
- A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.
- Searching becomes efficient due to the fact that now, the entire list will not be searched on every operation. Only hash table entries are checked using the key

and if an entry is found then the corresponding file will be fetched using the value.


File name 1
File name 2
Pointers to data blocks
Pointers to data blocks
Key 1
Value 1
Key 2
Value 2
Key n
Value n
Hash Table
Hash_Function(file_name) = key
Value
File