VMS operating system provides index and relative files

NFS
is standard UNIX client-server file sharing protocol
CIFS
is standard Windows protocol

File System Implementation

File System Structure

- The file system resides permanently on secondary storage which is designed to hold a large amount of data permanently
- Characteristics of disk which make them convenient medium for storing multiple files :-

1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same place.

2. A disk can access directly any block of information either sequentially or randomly, and switching from one file to another require minor movements of disk heads and rotational latency.

File system provide efficient and convenient access to the disk by allowing data to be stored, located and retrieved easily.

The file system itself is generally composed of many different levels.

Layered file System

Each level in the design uses features of lower levels to create new features for use by higher levels.

The layered approach to file system means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be filesystem specific.

- Common file systems in use include UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows system FAT, NTFS, CD-ROM systems etc.

- The features of layered file system are :-

1:> At the lowest layer are the physical devices, consisting of magnetic disk, motor and controls, and electronics connected to them and controlling them

2:> I/O control consist of device drivers and interrupt-handlers to transfer information between main memory and disk system.

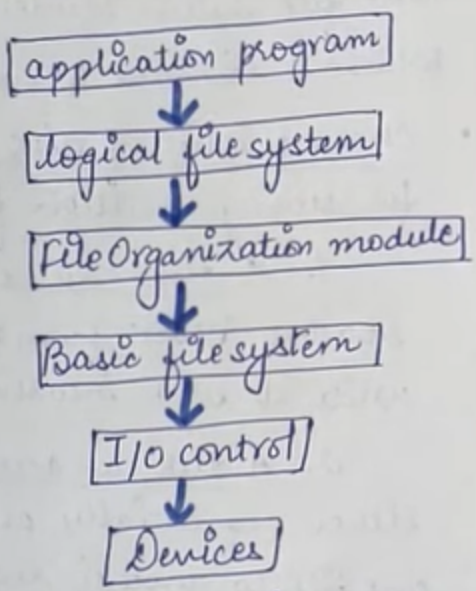


fig. layered file system.

- A device driver acts as a translator. Its input consists of high level command such as "read block 123". Its output consist of low level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O devices to the rest of the system.

3:> The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by numeric disk address (for example, drive 1, cylinder 73, track 2, sector 10).

- This layer also manages the memory buffers and caches that hold various file-system, directory, and data blocks

4.> The file organization module knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical blocks, the file organization module also maintains list of free blocks, and allocates free blocks to files as needed.

5.> The logical file system manages metadata information associated with a file except the data itself. This level manages the directory structure and the mapping of file names to file control block, FCB's which contain information about file including ownership, permissions, and location of the file contents i.e block number information for finding the data on the disk. It is also responsible for protection and security.

File System Implementation

- **Overview:**

→ Several on-disk and in-memory structures are used to implement a file system. These structures vary depending on the operating system and the file system but some general principles apply.

→ On-disk structures are described briefly :-

1. Boot Control Block :- contains information needed by system to boot OS from that volume.
   - Needed if volume contains OS, usually first block of volume.

HASH TABLE

- Here, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- Significantly decrease the search time.
- Deletion and Insertion is easy.

Difficulties are-

1. Generally fixed size and
2. Dependence of hash function on that size.

- In place, a chained overflow hash table can be used.
- Each hash entry can be a linked list instead of an individual value, and we can resolve collisions by adding the new entry to the linked list.
- Lookups may get slowed as they require linked lists for searching the name.

ALLOCATION METHODS

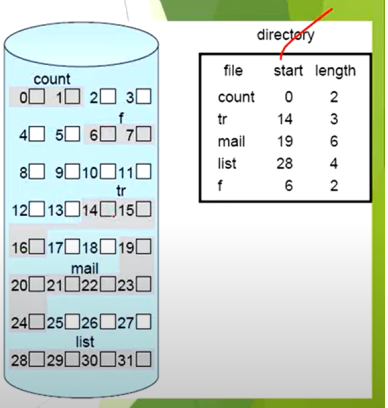
- The direct-access nature of disks allows us flexibility in the implementation of files.
- In almost every case, many files are stored on the same disk.
- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

CONTIGUOUS ALLOCATION

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk.
- Disk addresses define a linear ordering on the disk.
- With this ordering, assuming that only one job is accessing the disk, accessing block b + 1 after block b normally requires no head movement.
- When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next.
- Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.

CONTIGUOUS ALLOCATION

- As we had discussed in previous modules, contiguous allocation will suffer from fragmentation problems(external and internal)
- All these algorithms suffer from the problem of external fragmentation.
- As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks.
- For these problems, we need to deploy dynamic storage allocation techniques.

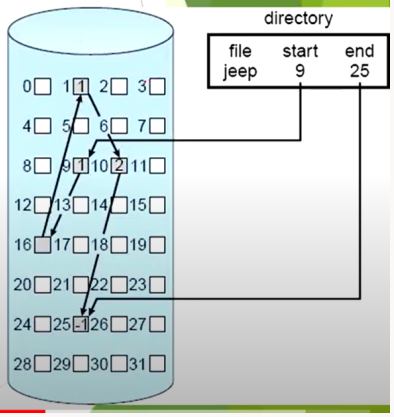


Contiguous allocation of disk space.

LINKED ALLOCATION

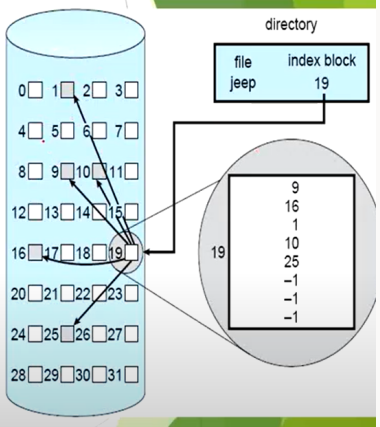
DISADVANTAGES

- Effective only for sequential-access files
- Space required for the pointers. If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.
- The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks.



INDEXED ALLOCATION

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation.
- In the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
- **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block**.
- The directory contains the address of the index.
- To find and read the ith block, we use the pointer in the ith index-block entry.



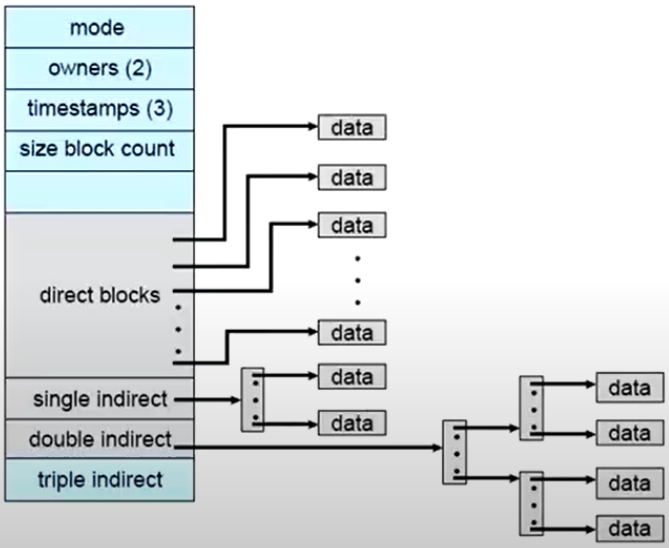
Indexed allocation of disk space.

INDEXED ALLOCATION

But how large the index block should be?

- Thus we need different schemes for indexing also.
  1. **Linked Scheme**-Index File in form of Linked List.
  2. **Multilevel Index**- Index File is stored in Multiple Levels.
  3. **Combined Scheme**-Here, the first, say, 15 pointers of the index block in the file's inode.
     - The first 12 of these pointers point to **direct blocks**; that is, they contain addresses of blocks that contain data of the file.
     - The next three pointers point to **indirect blocks**.
     - The first points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data.
     - The second points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks.
     - The last pointer contains the address of a **triple indirect block**.
- Suffers from performance issues just like linked allocation.

UNIX INODE


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

Free Space Management

- To keep track of free disk space, the system maintains a free space list.
- The free space list records all free disk blocks - those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to new file. This space is then removed from free-space list.
- When a file is deleted, its disk space is added to the free space list.
- There are 4 technique to implement free space management

1) Bit Vector

Each block is represented by 1 bit. If block is free bit is 1
If block is allocated then bit is 0.

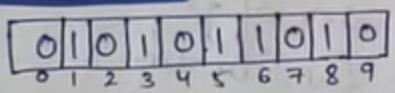


Fig. Bit Map or Bit Vector

Eg: There are 10 blocks, bit 0 represent block is allocated thus 0, 2, 4, 7, 9 blocks are allocated and bit 1 represents free block thus block 1, 3, 5, 6, 8 are free

Advantage:-

1. It is simple to implement and is efficient in finding the first free block or n consecutive free blocks on the disk.

Disadvantage:-

1. As entire bit vector need to be store in main memory for bit vector to be efficient. It is possible for smaller disk to keep bit map in main memory but with larger disk size the bit map size will also increase not possible to store in main memory

2) Linked List

- Another approach to free space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on disk and caching it in memory.

Advantage:

1. No wastage of space

Disadvantage:-

1. Pointer also needs space
2. Not efficient: to traverse the list, we must read each block, which requires substantial I/O time.

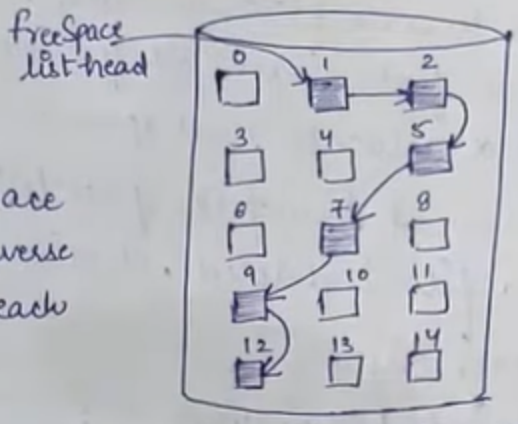


fig. Linked free-space list on disk.

3) Grouping

- It stores address of n-1 free blocks in 1st free block.

Advantage:-

- Addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

4) Counting

It is a modification in linked list approach, in addition to next free block pointer also maintain a variable 'count' saying how many blocks are contiguously free after the first block.

Efficiency and Performance

- Efficiency dependent on :
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache - separate section of main memory for frequently used blocks
  - free-behind and read-ahead - techniques to optimize sequential access.
  - improve PC performance by dedicating section of memory as virtual disk, & RAMdisk

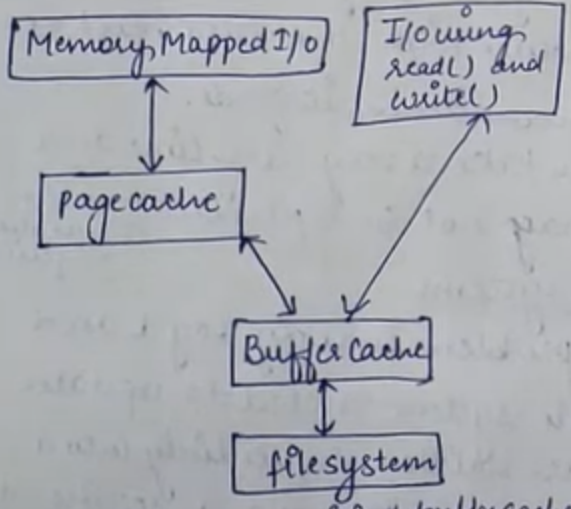


Fig1. I/O without a unified buffercache

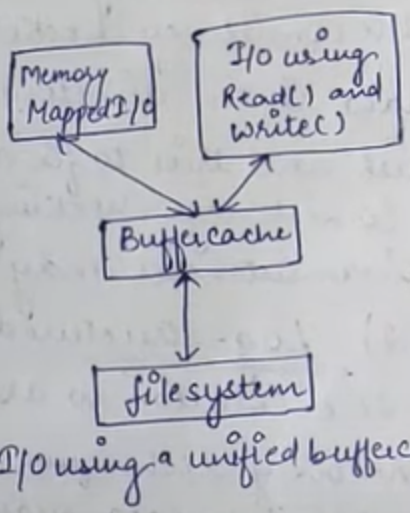


Fig2 I/O using a unified buffercache

- Buffer cache : It stores frequently accessed blocks
- Page cache : It stores pages accessed frequently

Fig1. has both page cache and buffer cache i.e double caching occurs.

Fig2. We have buffer cache only which store both accessed disk blocks and pages so that CPU can access quickly.

Recovery

- Files and directories are kept both in main memory and on disk, and care must be taken to ensure that a system failure does not result in loss of data or in data inconsistency.
- A system crash, bugs in file-system implementation, disk controllers and even user applications can corrupt a file system.
- There are various methods to deal with corruption, depending on file-system data structures and algorithms :-

1) Consistency Checking

- The consistency checker is a system program, compares the data in the directory structure with the data blocks on disk and tries to fix any inconsistencies it finds.
- Consistency checking can take a very long time and inconsistencies may or may not be reparable, human intervention required.

2) Log-Structured file systems

- The solution to above problem is apply log-based recovery techniques to file-system metadata updates.
- All metadata changes are written sequentially into a Log. Each set of operations for performing a specific task is a transaction.
- A transaction is considered comitted once it is written in the log, and the user process may continue executing.
- The transactions in the log are asynchronously written to the file system

- As the operations are completed, a pointer is updated to indicate which actions have completed and which are still incomplete
- When all operations are complete, the transaction is removed from the log file.
- If the file system crashes, all remaining transactions in the log must still be performed.

3) Back up and Restore

- In order to recover lost data in the event of disk crash, it is important to conduct backups regularly.
- files should be copied to some storage device, such as floppy disk, magnetic tape, optical disk or hard disk.
- Full backup copies every file on a filesystem.
- Incremental backup copy only files which have changed since some previous time.
- A combination of full and incremental back ups can offer a compromise between full recoverability, the number and size of backups tapes needed, and the number of tapes that need to be used to do a full restore
- Backup tapes are often reused, particularly for daily backups, but there are limits to how many times same tape can be used.
- Every so often a full backup should be made that is kept forever and not overwritten.