File System Management

File

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure

A File Structure should be according to a required format that the operating system can understand.

* A file has a certain defined structure according to its type.
* A text file is a sequence of characters organized into lines.
* A source file is a sequence of procedures and functions.
* An object file is a sequence of bytes organized into blocks that are understandable by the machine.
* When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

### Ordinary files

* These are the files that contain user information.
* These may have text, databases or executable program.
* The user can apply various operations on such files like add, modify, delete or even remove the entire file.

### Directory files

* These files contain list of file names and other information related to these files.

### Special files

* These files are also known as device files.
* These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

* **Character special files** − data is handled character by character as in case of terminals or printers.
* **Block special files** − data is handled in blocks as in the case of disks and tapes.

File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

* Sequential access
* Direct/Random access
* Indexed sequential access

### Sequential access

* Data is accessed one record right after another is an order.
* Read command cause a pointer to be moved ahead by one.
* Write command allocate space for the record and move the pointer to the new End Of File.
* Such a method is reasonable for tape.
* This access method is the most primitive one. Example: Compilers usually access files in this fashion.

### Direct/Random access

* Random access file organization provides, accessing the records directly.
* Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
* The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.
* This method is useful for disks.

### Indexed sequential access

* This mechanism is built up on base of sequential access.
* An index is created for each file which contains pointers to various blocks.
* Index is searched sequentially and its pointer is used to access the file directly.

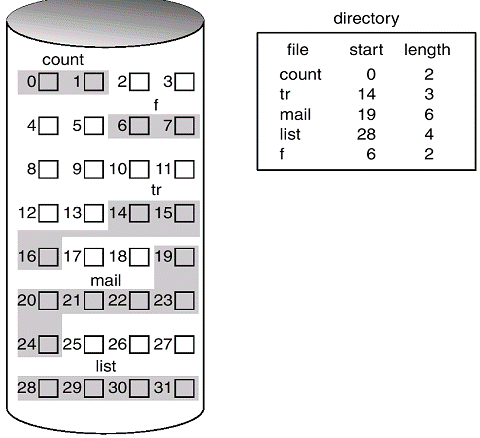
File Allocation Methods

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

* Contiguous Allocation
* Linked Allocation
* Indexed Allocation

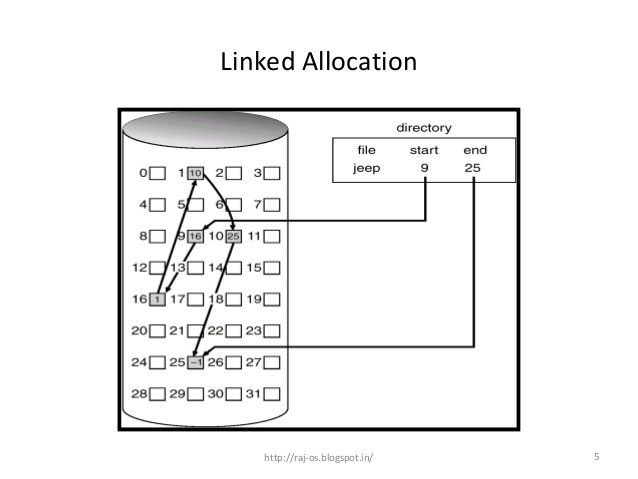
### Contiguous Allocation

* Each file occupies a contiguous address space on disk.
* Assigned disk address is in linear order.
* Easy to implement.
* External fragmentation is a major issue with this type of allocation technique.  As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists when enough total disk space exists to satisfy a request, but this space not contiguous; storage is fragmented into a large number of small holes.
* The difficulty with contiguous allocation is finding space for a new file. If the file to be created is n blocks long, then the OS must search for n free contiguous blocks. First-fit, best-fit, and worst-fit strategies are the most common strategies used to select a free hole from the set of available holes.



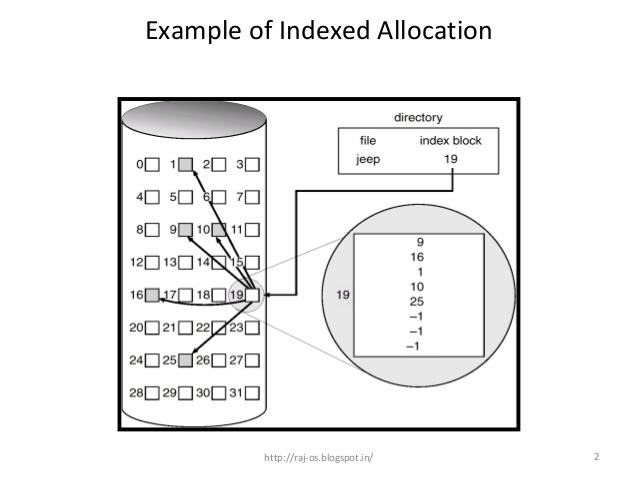
### Linked Allocation

* The problems in contiguous allocation can be traced directly to the requirement that the spaces be allocated contiguously and that the files that need these spaces are of different sizes. These requirements can be avoided by using linked allocation.
* Each file carries a list of links to disk blocks.
* The directory contains a pointer to the first and (optionally the last) block of the file. For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27. Each block contains a pointer to the next block and the last block contains a NIL pointer. The value -1 may be used for NIL to differentiate it from block 0.
* With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. A write to a file removes the first free block and writes to that block. This new block is then linked to the end of the file. To read a file, the pointers are just followed from block to block.
* No external fragmentation. Any free block can be used to satisfy a request. Notice also that there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks.
* Effectively used in sequential access file.
* Inefficient in case of direct access file. To find the ith block of a file, it must start at the beginning of that file and follow the pointers until the ith block is reached. Note that each access to a pointer requires a disk read.



### Indexed Allocation

* Provides solutions to problems of contiguous and linked allocation. This is done by bringing all the pointers together into one location called the index block. Of course, the index block will occupy some space.
* An index block is created having all pointers to files.
* Each file has its own index block which stores the addresses of disk space occupied by the file.
* Directory contains the addresses of index blocks of files.
* Indexed allocation supports direct access, without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.



Free Space Management

Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files. To keep track of free disk space, the system maintains a free-space list. The free-space list records all disk blocks that are free (i.e., are not allocated to some file). To create a file, the free-space list has to be searched for the required amount of space, and allocate that space to a new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

* Bit-Vector
* Linked List
* Grouping
* Counting

### Bit – Vector

* Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by a 1 bit. If the block is free, the bit is 0; if the block is allocated, the bit is 1. An index block is created having all pointers to files.
* For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free-space bit map would be:  
    
  11000011000000111001111110001111…
* The main advantage of this approach is that it is relatively simple and efficient to find n consecutive free blocks on the disk.
* Unfortunately, bit vectors are inefficient unless the entire vector is kept in memory for most accesses. Keeping it main memory is possible for smaller disks such as on microcomputers, but not for larger ones.

### Linked List

* Another approach is to link all the free disk blocks together, keeping a pointer to the first free block.

* This block contains a pointer to the next free disk block, and so on. In the previous example, a pointer could be kept to block 2, as the first free block.
* Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

### Grouping

* A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
* The first n-1 of these are actually free. The last one is the disk address of another block containing addresses of another n free blocks.
* The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

### Counting

* Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when contiguous allocation is used.
* Thus, rather than keeping a list of free disk addresses, the address of the first free block is kept and the number n of free contiguous blocks that follow the first block.
* Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

Virtual File System Manager

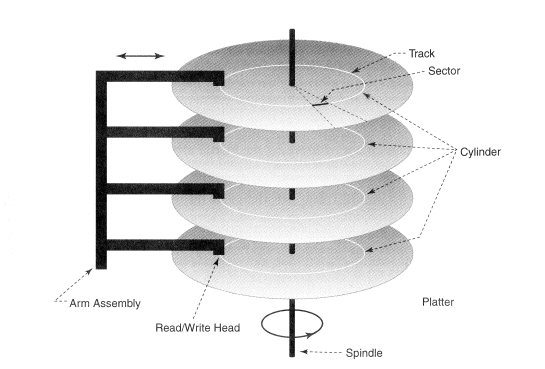
A virtual file system (VFS) is programming that forms an interface between an operating system's [kernel](http://searchenterpriselinux.techtarget.com/definition/kernel) and a more concrete [file system](http://searchstorage.techtarget.com/definition/file-system).

The VFS serves as an abstraction layer that gives applications access to different types of file systems and local and network storage devices.

For that reason, a VFS may also be known as a virtual file system switch or virtual file system manager. It also manages the data storage and retrieval between the operating system and the storage sub-system.

The VFS maintains a [cache](http://searchstorage.techtarget.com/definition/cache) of [directory](http://searchwinit.techtarget.com/definition/directory) lookups to enable easy location of frequently accessed directories.

Disk Structure



* Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
* The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.

– Sector 0 is the first sector of the first track on the outermost cylinder.

– Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Disk Scheduling

* The operating system is responsible for using hardware efficiently - for the disk drivers, this means having a fast access time. The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
* Access time has two major components.
  + **Seek time** is the time that the disk moves the heads to the cylinder containing the desired sector. Typically 5-10 milliseconds.
  + **Rotational latency** is an additional time waiting for the disk to rotate the desired sector to the disk head. Typically 2-4 milliseconds.
* Should ensure a fast access time and disk bandwidth
* Minimize seek time.
* Seek time ≈ seek distance
* Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling Algorithms

Several algorithms exist to schedule the servicing of disk I/O requests.

* FCFS
* SSTF
* SCAN
* C-SCAN
* C-LOOK

### FCFS

* Service requests in the order they come.
* Fair to all requests.
* Can cause very large total seek time over all requests if the load is moderate to high.

### SSTF

* Shortest Seek Time First - Selects the request with the minimum seek time from the current head position.
* SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
* Minimizes seek time but not fair.
* May work well if the load is not high.

### SCAN

* The disk arm starts at one end of the disk and moves toward the other end, servicing requests until it will get to the other end of the disk, where the head movement is reversed and the servicing continues.
* Sometimes called the elevator algorithm.

### C-SCAN

* Provides a more uniform wait time than SCAN.
* The head moves from one end of the disk to the other. Servicing requests as it goes. However, when it reaches the other end, it immediately will return to the beginning of the disk, without servicing any requests on the return trip.
* Treats the cylinders as a wraparound circular list from the first cylinder to the last one.

### C-LOOK

* A version of C-SCAN.
* Arm goes only as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
* Similarly, LOOK is a version of SCAN which only goes as far as the last request in each direction.

Selecting a Disk Scheduling Algorithm

* SSTF is common and has a natural appeal.
* SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
* Performance depends on the number and types of requests.
* Requests for disk service can be influenced by the file allocation method.

Disk Management

* Low-level formatting, or physical formatting - Dividing a disk into sectors that the disk controller can read and write.
* To use a disk to hold files, the operating system still needs to record its own data structures on the disk.

– Partition the disk into one or more groups of cylinders.

– Logical formatting or “making a file system” on those partitions.

* Boot block initializes system.

– The bootstrap is stored in ROM.

– Bootstrap loads the boot program from the boot block.

– The boot block can be on the disk, a floppy diskette or a CD-ROM.

* Methods such as sector sparing used to handle bad blocks.

Data Striping

* Data Striping is a method of concatenating multiple drives into one logical storage unit. Striping involves partitioning each drive's storage space into stripes. These stripes are then interleaved, so that the combined space is considered as one drive.
* Most multi-user operating systems today, like Unix, Windows2000 and Netware, support overlapped disk I/O operations across multiple drives. However, in order to maximize throughput for the disk subsystem, the I/O load must be balanced across all the drives so that each drive can be kept busy as much as possible. In a multiple drive system without striping, the disk I/O load is never perfectly balanced. Some drives will contain data files which are frequently accessed and some drives will only rarely be accessed.