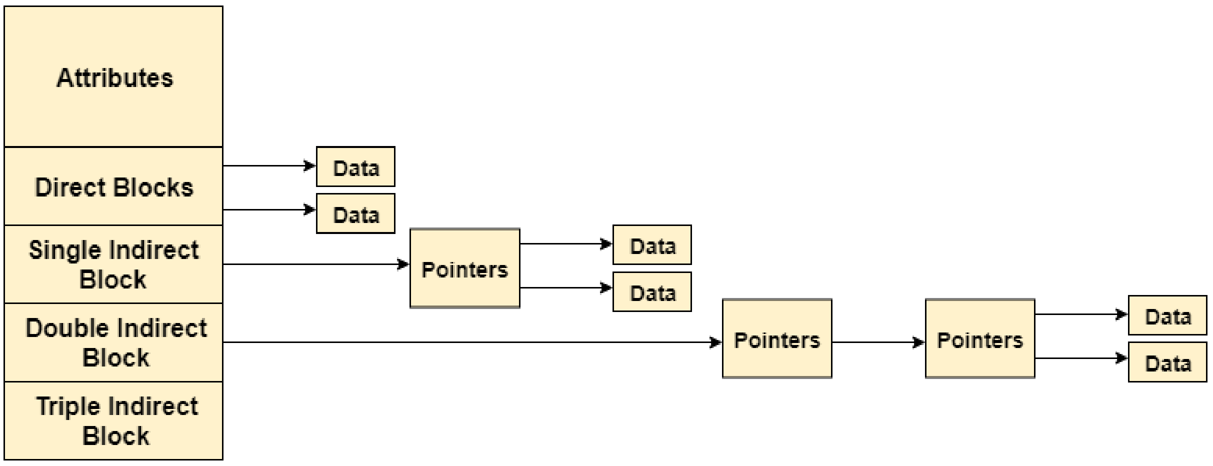
UNIX iNode Structure

In UNIX based operating systems, each file is indexed by an Inode. Inode are the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of Inodes in the file system.

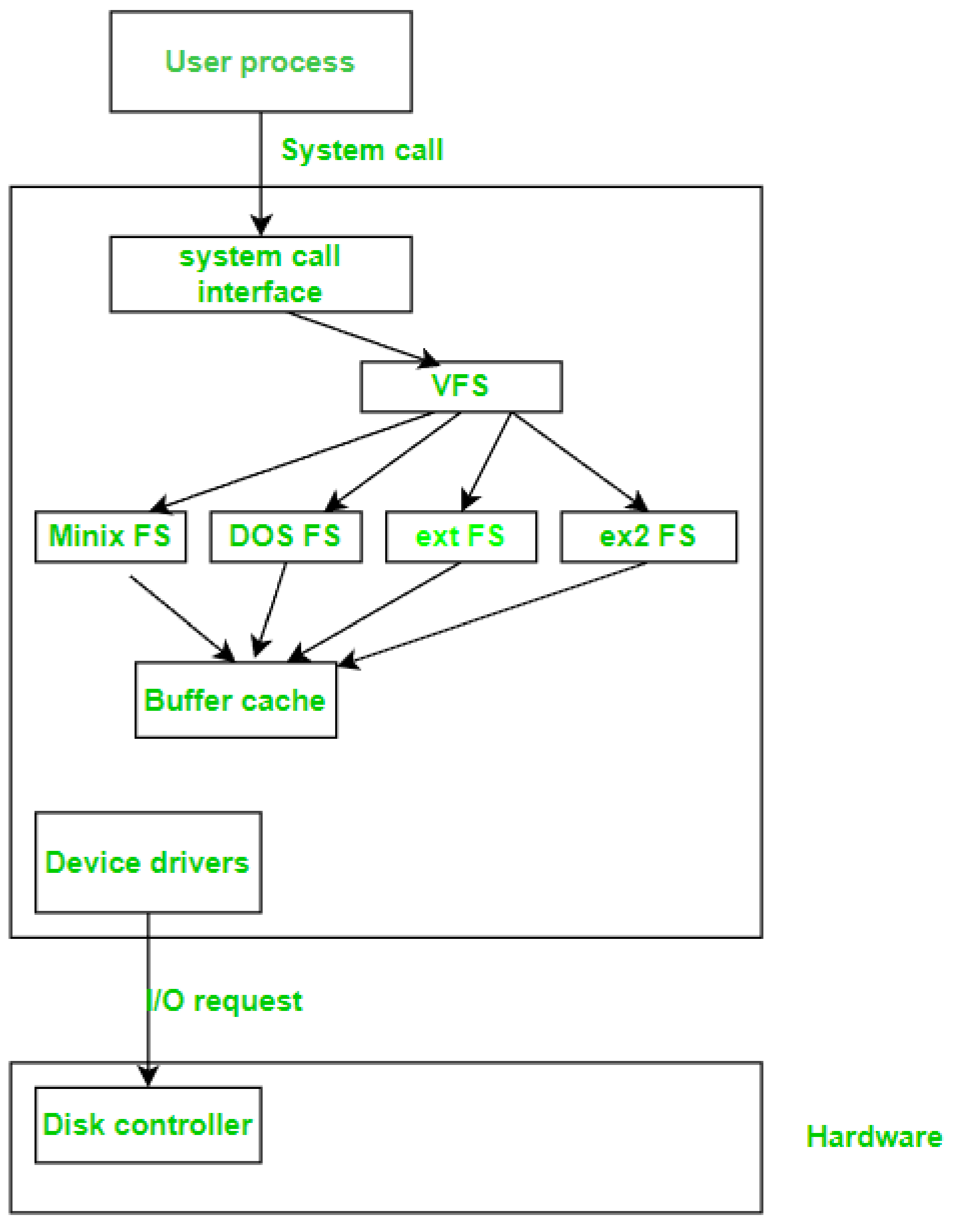
An Inode includes the following information

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.



Linux Virtual File System

The virtual file system is one of the best features of Linux that makes Linux an operating system of choice.



Virtual File System

As shown in the diagram:
The user process initiates the execution of the system call with the help of the VFS method. VFS interprets this stimulus into an appropriate internal file system call which in turn activates the concerned mapping function of the specified file system.

In the case of an operating system that demands structural changes, all required constructs are created dynamically to meet the requirements. Ex: In FAT-compatible systems, directories are not treated as files so such characteristics are dynamically created since they cannot be mapped. The target file system is then activated to take control so that results can be delivered to the user of the system.

As seen in the above diagram **actions** and **reactions** are observed as a role of VFS:
The user initiates a file-oriented system call.

1. Kernel calls function in VFS.
2. Execution of file system independent manipulation and initiation of target file system call.
3. mapping of calls from VFS to the target file system.
4. The target file system converts the request into device driver execution.
5. Appropriate action is completed as per the guidance of the device driver function and the results are obtained.
6. Since VFS objects are implemented as C data structures. Every object is attached with data and pointers.