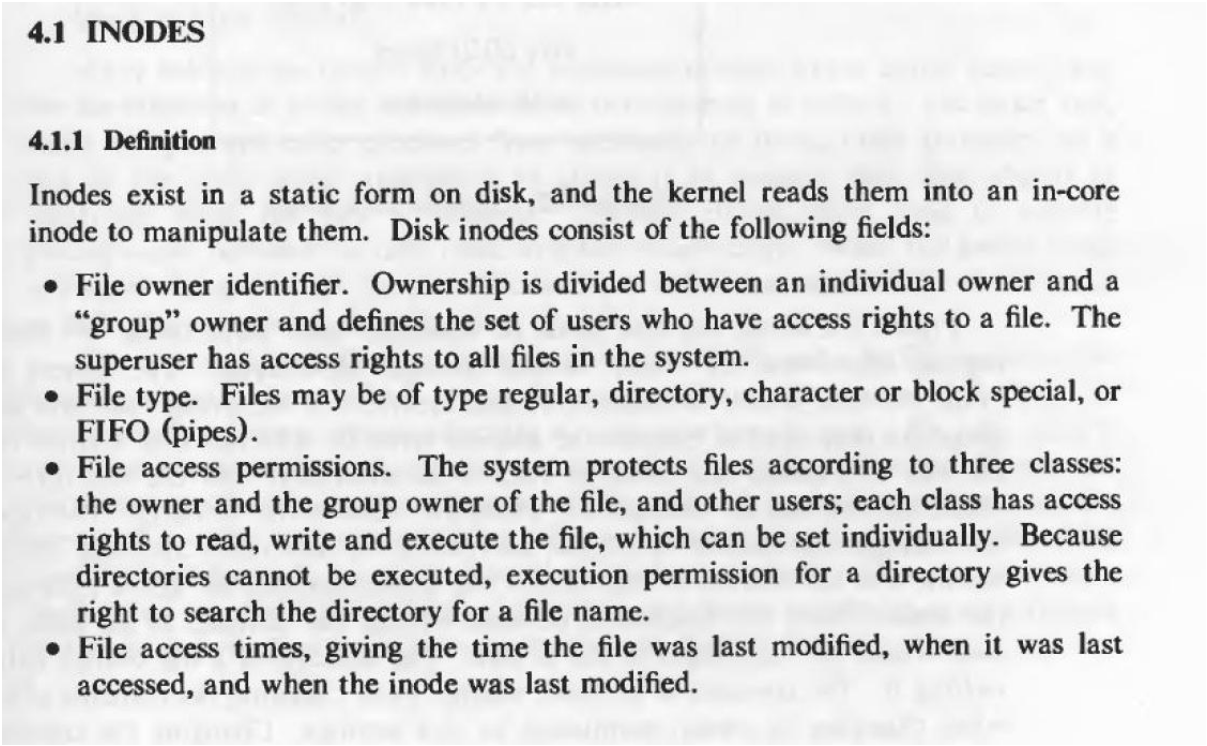
File Management:

Contents :

1. File operations and file system calls
2. File system software architecture
3. Unix inode-
4. File Organization and Access – sequential, index-sequential
5. File allocation- contiguous, non-contiguous, linked, indexed – unix VFS diagram
6. File Directories –directory operations, single level directory, two level directory, tree structure directory, acyclic graph directory, file system mounting – mount/umount
7. File directory information
8. File Sharing – access rights and management
9. Secondary Storage Management - FAT
10. Linux Virtual File System

Unix inode: (From The design of Unix OS by Maurice Bach)



4.1 INODES

4.1.1 Definition

Inodes exist in a static form on disk, and the kernel reads them into an in-core inode to manipulate them. Disk inodes consist of the following fields:

- File owner identifier. Ownership is divided between an individual owner and a "group" owner and defines the set of users who have access rights to a file. The superuser has access rights to all files in the system.
- File type. Files may be of type regular, directory, character or block special, or FIFO (pipes).
- File access permissions. The system protects files according to three classes: the owner and the group owner of the file, and other users; each class has access rights to read, write and execute the file, which can be set individually. Because directories cannot be executed, execution permission for a directory gives the right to search the directory for a file name.
- File access times, giving the time the file was last modified, when it was last accessed, and when the inode was last modified.

- Number of links to the file, representing the number of names the file has in the directory hierarchy. Chapter 5 explains file links in detail.
- Table of contents for the disk addresses of data in a file. Although users treat the data in a file as a logical stream of bytes, the kernel saves the data in discontiguous disk blocks. The inode identifies the disk blocks that contain the file's data.
- File size. Data in a file is addressable by the number of bytes from the beginning of the file, starting from byte offset 0, and the file size is 1 greater than the highest byte offset of data in the file. For example, if a user creates a file and writes only 1 byte of data at byte offset 1000 in the file, the size of the file is 1001 bytes.

The inode does not specify the path name(s) that access the file.

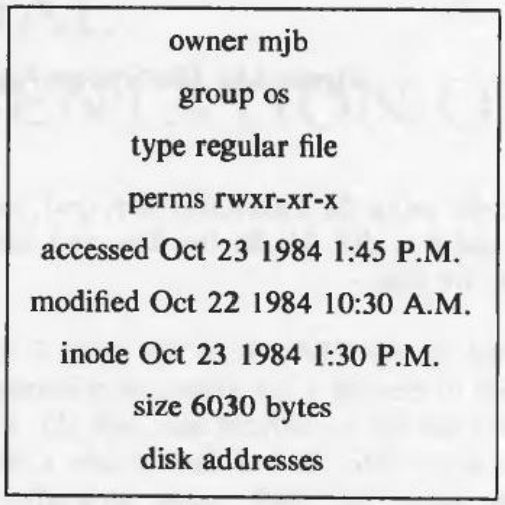


Figure 4.2. Sample Disk Inode

Figure 4.2 shows the disk inode of a sample file. This inode is that of a regular file owned by "mjb," which contains 6030 bytes. The system permits "mjb" to read, write, or execute the file; members of the group "os" and all other users can only read or execute the file, not write it. The last time anyone read the file was on October 23, 1984, at 1:45 in the afternoon, and the last time anyone wrote the file was on October 22, 1984, at 10:30 in the morning. The inode was last changed on October 23, 1984, at 1:30 in the afternoon, although the data in the file was not written at that time. The kernel encodes the above information in the inode. Note the distinction between writing the contents of an inode to disk and writing the contents of a file to disk. The contents of a file change only when *writ*ing it. The contents of an inode change when changing the contents of a file or when changing its owner, permission, or link settings. Changing the contents of a

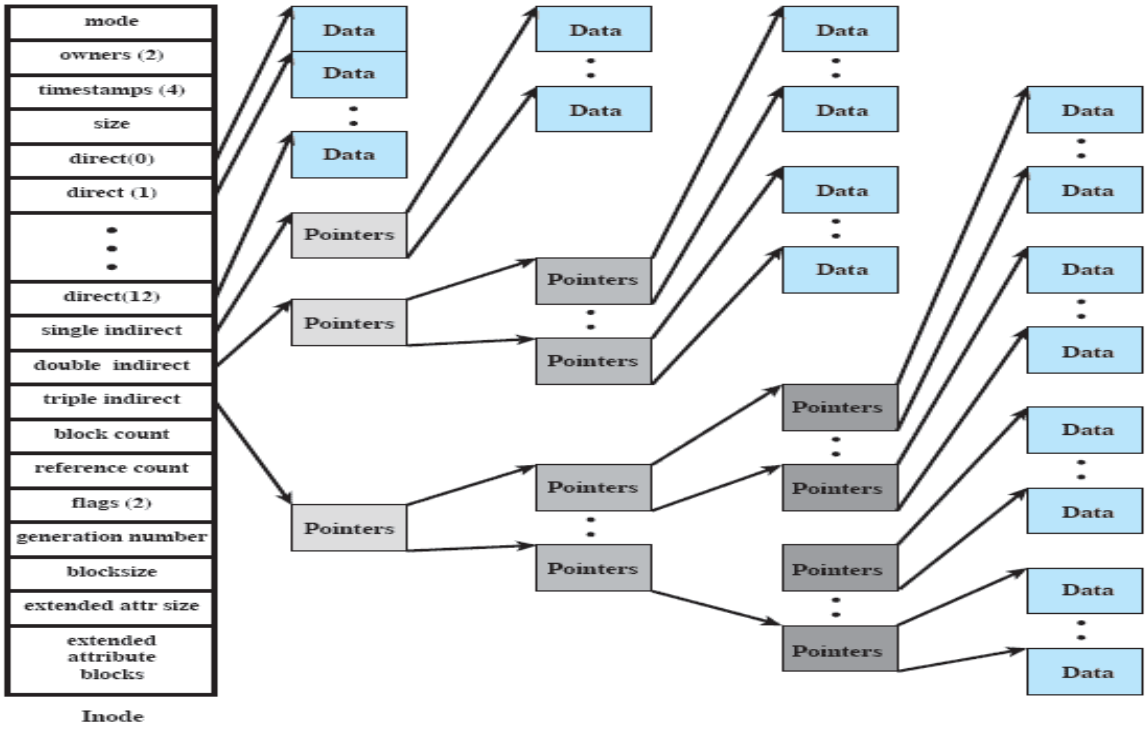
file automatically implies a change to the inode, but changing the inode does not imply that the contents of the file change.

The in-core copy of the inode contains the following fields in addition to the fields of the disk inode:

- The status of the in-core inode, indicating whether
  - the inode is locked,
  - a process is waiting for the inode to become unlocked,
  - the in-core representation of the inode differs from the disk copy as a result of a change to the data in the inode,
  - the in-core representation of the file differs from the disk copy as a result of a change to the file data,
  - the file is a mount point (Section 5.15).
- The logical device number of the file system that contains the file.
- The inode number. Since inodes are stored in a linear array on disk (recall Section 2.2.1), the kernel identifies the number of a disk inode by its position in the array. The disk inode does not need this field.
- Pointers to other in-core inodes. The kernel links inodes on hash queues and on a free list in the same way that it links buffers on buffer hash queues and on the buffer free list. A hash queue is identified according to the inode's logical device number and inode number. The kernel can contain at most one in-core copy of a disk inode, but inodes can be simultaneously on a hash queue and on the free list.
- A reference count, indicating the number of instances of the file that are active (such as when *open*ed).

Many fields in the in-core inode are analogous to fields in the buffer header, and the management of inodes is similar to the management of buffers. The inode lock, when set, prevents other processes from accessing the inode; other processes set a flag in the inode when attempting to access it to indicate that they should be awakened when the lock is released. The kernel sets other flags to indicate discrepancies between the disk inode and the in-core copy. When the kernel needs to record changes to the file or to the inode, it writes the in-core copy of the inode to disk after examining these flags.

The most striking difference between an in-core inode and a buffer header is the in-core reference count, which counts the number of active instances of the file. An inode is active when a process allocates it, such as when *open*ing a file. An inode is on the free list only if its reference count is 0, meaning that the kernel can reallocate the in-core inode to another disk inode. The free list of inodes thus serves as a cache of inactive inodes: If a process attempts to access a file whose inode is not currently in the in-core inode pool, the kernel reallocates an in-core inode from the free list for its use. On the other hand, a buffer has no reference count; it is on the free list if and only if it is unlocked.



10 direct blocks with 1K bytes each =	10K bytes
1 indirect block with 256 direct blocks =	256K bytes
1 double indirect block with 256 indirect blocks =	64M bytes
1 triple indirect block with 256 double indirect blocks =	16G bytes

Figure 4.7. Byte Capacity of a File — 1K Bytes Per Block

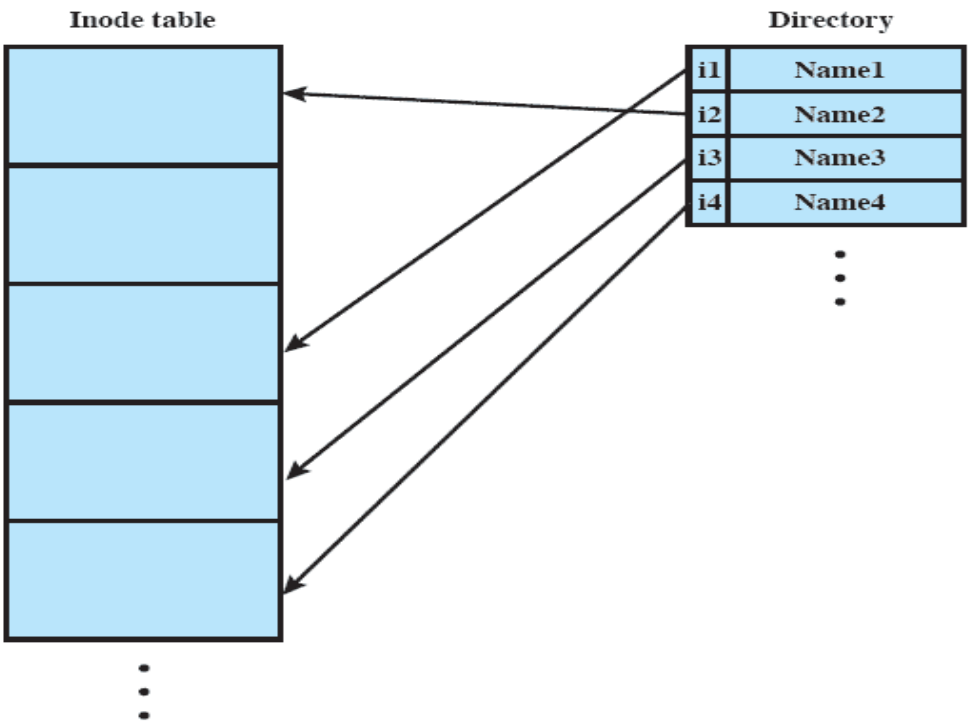


Figure 12.15 UNIX Directories and Inodes

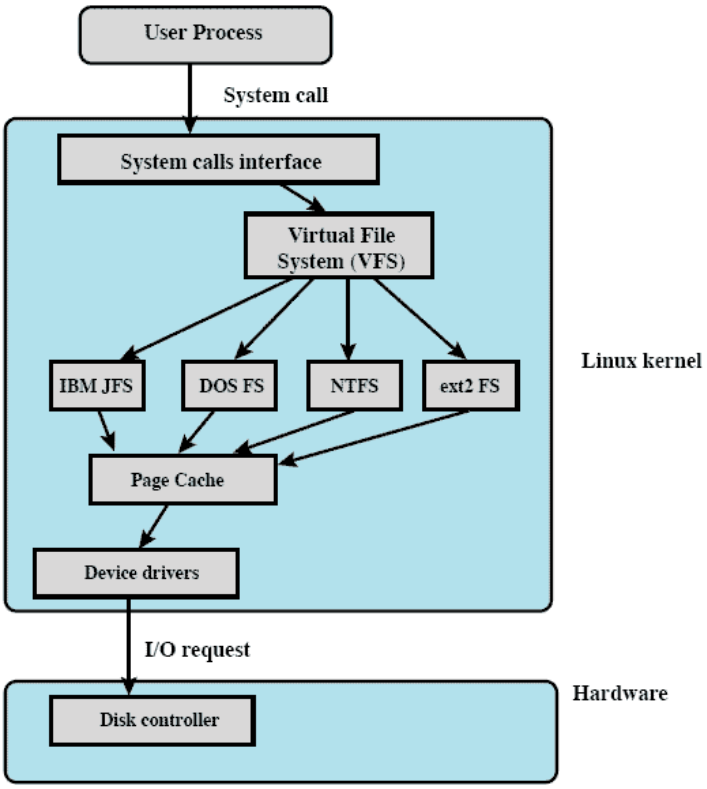
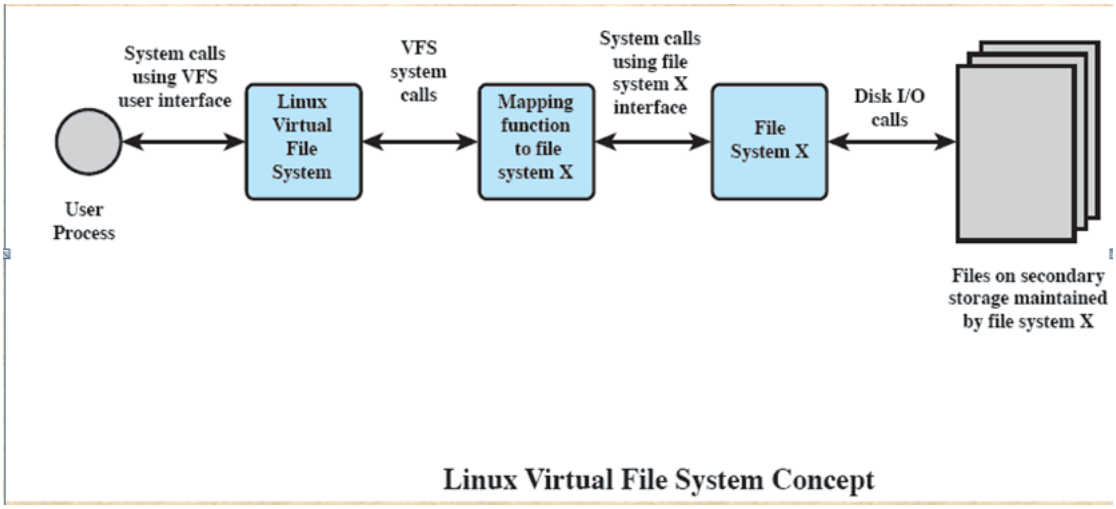


Figure 12.17 Linux Virtual File System Context

Role of VFS within kernel

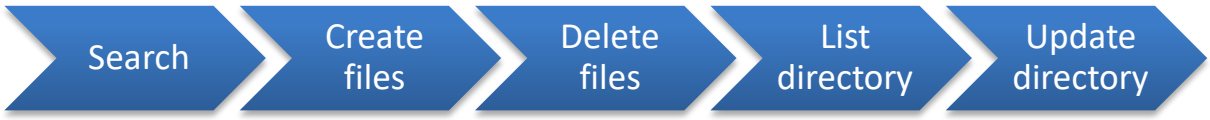


Linux Virtual File System Concept

File directory information:

	Basic Information
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
	Address Information
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
	Access Control Information
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
	Usage Information
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Operations on directory :



File allocation methods:

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium