inode

The **inode** (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory. Each inode stores the attributes and disk block locations of the object's data.[1] File-system object attributes may include metadata (times of last change,[2] access, modification), as well as owner and permission data.[3]

Directories are lists of names assigned to inodes. A directory contains an entry for itself, its parent, and each of its children.

Contents

Etymology

There has been uncertainty on the Linux kernel mailing list about the reason for the "i" in "inode". In 2002, the question was brought to Unix pioneer Dennis Ritchie, who replied:[4]

> In truth, I don't know either. It was just a term that we started to use. "Index" is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk, with all the hierarchical directory information living aside from this. Thus the i-number is an index in this array, the i-node is the selected element of the array. (The "i-" notation was used in the 1st edition manual; its hyphen was gradually dropped.)

A 1978 paper by Ritchie and Ken Thompson bolsters the notion of "index" being the etymological origin of inodes. They wrote:[5]

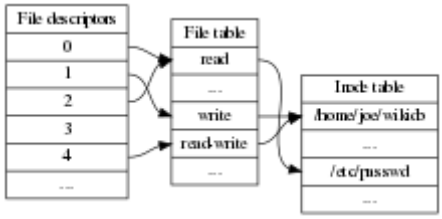
> […] a directory entry contains only a name for the associated file and a pointer to the file itself. This pointer is an integer called the *i-number* (for index number) of the file. When the file is accessed, its i-number is used as an index into a system table (the *i-list*) stored in a known part

> of the device on which the directory resides. The entry found thereby (the file's *i-node*) contains the description of the file.

Additionally, Maurice J. Bach wrote that an inode "is a contraction of the term index node and is commonly used in literature on the UNIX system".[6]

Details

A file system relies on data structures *about* the files, as opposed to the contents of that file. The former are called *metadata*—data that describes data. Each file is associated with an *inode*, which is identified by an integer, often referred to as an *i-number* or *inode number*.



File descriptors, file table and inode table in Unix[7]

Inodes store information about files and directories (folders), such as file ownership, access mode (read, write, execute permissions), and file type. On many older file system implementations, the maximum number of inodes is fixed at file system creation, limiting the maximum number of files the file system can hold. A typical allocation heuristic for inodes in a file system is one inode for every 2K bytes contained in the filesystem.[8]

The inode number indexes a table of inodes in a known location on the device. From the inode number, the kernel's file system driver can access the inode contents, including the location of the file, thereby allowing access to the file. A file's inode number can be found using the `ls -i` command. The `ls -i` command prints the i-node number in the first column of the report.

Some Unix-style file systems such as ReiserFS, btrfs, and APFS omit a fixed-size inode table, but must store equivalent data in order to provide equivalent capabilities. The data may be called stat data, in reference to the `stat` system call that provides the data to programs. Common alternatives to the fixed-size table include B-trees and the derived B+ trees.

File names and directory implications:

- Inodes do not contain its hardlink names, only other file metadata.
- Unix directories are lists of association structures, each of which contains one filename and one inode number.
- The file system driver must search a directory looking for a particular filename and then convert the filename to the correct corresponding inode number.

The operating system kernel's in-memory representation of this data is called `struct inode` in Linux. Systems derived from BSD use the term `vnode` (the "v" refers to the kernel's virtual file system layer).

POSIX inode description

The POSIX standard mandates file-system behavior that is strongly influenced by traditional UNIX file systems. An inode is denoted by the phrase "file serial number", defined as a *per-file system* unique identifier for a file.[9] That file serial number, together with the device ID of the device containing the file, uniquely identify the file within the whole system.[10]

Within a POSIX system, a file has the following attributes[10] which may be retrieved by the `stat` system call:

- Device ID (this identifies the device containing the file; that is, the scope of uniqueness of the serial number).
- File serial numbers.
- The file *mode* which determines the file type and how the file's owner, its group, and others can access the file.
- A link count telling how many hard links point to the inode.
- The User ID of the file's owner.
- The Group ID of the file.
- The device ID of the file if it is a device file.
- The size of the file in bytes.
- Timestamps telling when the inode itself was last modified (`ctime`, *inode change time*), the file content last modified (`mtime`, *modification time*), and last accessed (`atime`, *access time*).
- The preferred I/O block size.
- The number of blocks allocated to this file.

Implications

- Files can have multiple names. If multiple names hard link to the same inode then the names are equivalent; i.e., the first to be created has no special status. This is unlike symbolic links, which depend on the original name, not the inode (number).
- An inode may have no links. An unlinked file is removed from disk, and its resources are freed for reallocation but deletion must wait until all processes that have opened it finish accessing it. This includes executable files which are implicitly held open by the processes executing them.
- It is typically not possible to map from an open file to the filename that was used to open it. The operating system immediately converts the filename to an inode number then discards the filename. This means that the `getcwd()` and `getwd()` library functions search the parent directory to find a file with an inode matching the working directory, then search that directory's parent, and so on until reaching the root directory. SVR4 and Linux systems maintain extra information to make this possible.
- Historically, it was possible to hard link directories. This made the directory structure into an arbitrary directed graph contrary to a directed acyclic graph. It was even possible for a directory to be its own parent. Modern systems generally prohibit this confusing state, except that the parent of *root* is still defined as root. The most notable exception to this prohibition is found in Mac OS X (versions 10.5 and higher) which allows hard links of directories to be created by the superuser.[11]
- A file's inode number stays the same when it is moved to another directory on the same device, or when the disk is defragmented which may change its physical location. This also implies that completely conforming inode behavior is impossible to implement with many non-Unix file systems, such as FAT and its descendants, which don't have a way of storing this invariance when both a file's directory entry and its data are moved around.
- Installation of new libraries is simple with inode file systems. A running process can access a library file while another process replaces that file, creating a new inode, and an all-new mapping will exist for the new file so that subsequent attempts to access the library get the new version. This facility eliminates the need to reboot to replace currently mapped libraries.
- It is possible for a device to run out of inodes. When this happens, new files cannot be created on the device, even though there may be free space available. This is most common for use cases like mail servers which contain many small files. File systems (such as JFS or XFS)

escape this limitation with extents or dynamic inode allocation, which can "grow" the file system or increase the number of inodes.

Inlining

It can make sense to store very small files in the inode itself to save both space (no data block needed) and lookup time (no further disk access needed). This file system feature is called inlining. The strict separation of inode and file data thus can no longer be assumed when using modern file systems.

If the data of a file fits in the space allocated for pointers to the data, this space can conveniently be used. For example, ext2 and its successors store the data of symlinks (typically file names) in this way if the data is no more than 60 bytes ("fast symbolic links").[12]

Ext4 has a file system option called `inline_data` that allows ext4 to perform inlining if enabled during file system creation. Because an inode's size is limited, this only works for very small files.[13]

In non-Unix systems

- NTFS has a main file table (MFT) storing files in a B-tree. Each entry has a "fileID", anologous to the inode number, that uniquely refers to this entry.[14] The three timestamps, a device ID, attributes, reference count, and file sizes are found in the entry, but unlike in POSIX the permissions are expressed through a different API.[15] The on-disk layout is more complex.[16] The earlier FAT file systems did not have such a table and were incapable of making hard links.
- The same stat-like `GetFileInformationByHandle` API can be used on ReFS, Cluster Shared Volumes, and SMB 3.0, so these systems presumably have a similar concept of a file ID. ReFS has a 128-bit file ID; this extension was also backported to NTFS, which originally had a 64-bit file ID.[15]

See also

- inode pointer structure
- inotify

References

1. Tanenbaum, Andrew S. *Modern Operating Systems* (3rd ed.). p. 279.
2. JVSANTEN. "Difference between mtime, ctime and atime - Linux Howtos and FAQs" (http://www.linux-faqs.info/general/difference-between-mtime-ctime-and-atime). *Linux Howtos and FAQs*.
3. "Anatomy of the Linux virtual file system switch" (http://www.ibm.com/developerworks/library/l-virtual-filesystem-switch). *ibm.com*.
4. Linux Kernel list archive (http://lkml.indiana.edu/hypermail/linux/kernel/0207.2/1182.html). Retrieved on 2011-01-12.
5. Ritchie, Dennis M.; Thompson, Ken (1978). "The UNIX Time-Sharing System" (https://archive.org/details/bstj57-6-1905). *The Bell System Technical Journal*. **57** (6): 1913–1914. Retrieved 19 December 2015.
6. Maurice J. Bach (1986). *The Design of the UNIX Operating System* (https://www.amazon.com/Design-Operating-System-Prentice-Hall-Software/dp/0132017997). Prentice Hall. ISBN 978-0132017992.

7. Bach, Maurice J. (1986). *The Design of the UNIX Operating System*. Prentice Hall. p. 94.
8. "linfo" (http://www.linfo.org/inode.html). *The Linux Information Project*. Retrieved 11 March 2020.
9. "Definitions - 3.176 File Serial Number" (http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_176). *The Open Group*. Retrieved 10 January 2018.
10. "<sys/stat.h>" (http://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/stat.h.html). *The Open Group*. Retrieved 15 January 2018.
11. "What is the Unix command to create a hardlink to a directory in OS X?" (https://stackoverflow.com/questions/80875/what-is-the-unix-command-to-create-a-hardlink-to-a-directory-in-os-x). *Stack Overflow*. 16 Jan 2011. Archived (https://web.archive.org/web/20200105174407/https://stackoverflow.com/questions/80875/what-is-the-unix-command-to-create-a-hardlink-to-a-directory-in-os-x) from the original on 5 January 2020. Retrieved 5 Jan 2020.
12. "The Linux kernel: Filesystems" (http://www.win.tue.nl/~aeb/linux/lk/lk-7.html). *tue.nl*.
13. "Ext4 Disk Layout" (https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Inline_Data). *kernel.org*. Retrieved August 18, 2013.
14. "Does Windows have Inode Numbers like Linux?" (https://stackoverflow.com/a/42475374). *Stack Overflow*.
15. "GetFileInformationByHandle function (fileapi.h) - Win32 apps" (https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getfileinformationbyhandle). *docs.microsoft.com*.
16. "[MS-FSCC]: NTFS Attribute Types" (https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/a82e9105-2405-4e37-b2c3-28c773902d85). *docs.microsoft.com*.

External links

- Anatomy of the Linux File System (https://web.archive.org/web/20150509080554/http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/)
- Inode definition (http://www.linfo.org/inode.html)
- Explanation of Inodes, Symlinks, and Hardlinks (http://linuxgazette.net/105/pitcher.html)
