Hierarchical File System

Hierarchical File System (**HFS**) is a proprietary file system developed by Apple Inc. for use in computer systems running Mac OS. Originally designed for use on floppy and hard disks, it can also be found on read-only media such as CD-ROMs. HFS is also referred to as **Mac OS Standard** (or "HFS Standard"), while its successor, HFS Plus, is also called *Mac OS Extended* (or "HFS Extended").

With the introduction of Mac OS X 10.6, Apple dropped support for formatting or writing HFS disks and images, which remain supported as read-only volumes.[1] Starting with macOS 10.15, HFS disks can no longer be read.

Contents

History

Apple introduced HFS in September 1985, specifically to support Apple's first hard disk drive for the Macintosh, replacing the Macintosh File System (MFS), the original file system which had been introduced over a year and a half earlier with the first Macintosh computer. HFS drew heavily upon Apple's first hierarchical operating system (SOS) for the failed Apple III, which also served as the basis for hierarchical file systems on the Apple IIe and Apple Lisa. HFS was developed by Patrick Dirks and Bill Bruffey. It shared a number of design features with MFS that were not available in other file systems of the time (such as DOS's FAT). Files could have multiple forks (normally a data and a resource fork), which allowed the main data of the file to be stored separately from resources such as icons that might need to be localized. Files were referenced with unique file IDs rather than file names, and file names could be 255 characters long (although the Finder only supported a maximum of 31 characters).

However, MFS had been optimized to be used on very small and slow media, namely floppy disks, so HFS was introduced to overcome some of the performance problems that arrived with the

HFS

Developer(s)	Apple Computer
Full name	Hierarchical File System
Introduced	September 17, 1985 with System 2.1
Partition identifier	`Apple_HFS` (Apple Partition Map) `0xAF` (MBR) *HFS and HFS+*
Structures	
Directory contents	B-tree
File allocation	Bitmap
Bad blocks	B-tree
Limits	
Max. volume size	2 TB (2×1024^4 bytes)
Max. file size	2 GB (2×1024^3 bytes)
Max. number of files	65535
Max. filename length	31 characters
Allowed characters in filenames	All 8-bit values except colon ":". Discouraged null and nonprints.
Features	
Dates recorded	Creation, modification, backup
Date range	January 1, 1904 - February 6, 2040
Date	1s

introduction of larger media, notably hard drives. The main concern was the time needed to display the contents of a folder. Under MFS all of the file and directory listing information was stored in a single file, which the system had to search to build a list of the files stored in a particular folder. This worked well with a system with a few hundred kilobytes of storage and perhaps a hundred files, but as the systems grew into megabytes and thousands of files, the performance degraded rapidly.

The solution was to replace MFS's directory structure with one more suitable to larger file systems. HFS replaced the flat table structure with the *Catalog File* which uses a B-tree structure that could be searched very quickly regardless of size. HFS also redesigned various structures to be able to hold larger numbers, 16-bit integers being replaced by 32-bit almost universally. Oddly, one of the few places this "upsizing" did not take place was the file directory itself, which limits HFS to a total of 65,535 files on each logical disk.

While HFS is a proprietary file system format, it is well-documented; there are usually solutions available to access HFS-formatted disks from most modern operating systems.

Apple introduced HFS out of necessity with its first 20 MB hard disk offering for the Macintosh in September 1985, where it was loaded into RAM from a MFS floppy disk on boot using a patch file ("Hard Disk 20"). However, HFS was not widely introduced until it was included in the 128K ROM that debuted with the Macintosh Plus in January 1986 along with the larger 800 KB floppy disk drive for the Macintosh that also used HFS. The introduction of HFS was the first advancement by Apple to leave a Macintosh computer model behind: the original 128K Macintosh, which lacked sufficient memory to load the HFS code and was promptly discontinued.

resolution	
Forks	Only 2 (data and resource)
Attributes	Color (3 bits, all other flags 1 bit), locked, custom icon, bundle, invisible, alias, system, stationery, inited, no INIT resources, shared, desktop
File system permissions	AppleShare
Transparent compression	Yes (third-party), Stacker
Transparent encryption	No
Other	
Supported operating systems	Classic Mac OS, macOS, Linux, Microsoft Windows (through MacDrive or Boot CampIFS drivers)

In 1998, Apple introduced HFS Plus to address inefficient allocation of disk space in HFS and to add other improvements. HFS is still supported by current versions of Mac OS, but starting with Mac OS X, an HFS volume cannot be used for booting, and beginning with Mac OS X 10.6 (Snow Leopard), HFS volumes are read-only and cannot be created or updated. In macOS Sierra (10.12), Apple's release notes state that "The HFS Standard filesystem is no longer supported."[2] However, read-only HFS Standard support is still present in Sierra and works as it did in previous versions.

Design

A storage volume is inherently divided into *logical blocks* of 512 bytes. The Hierarchical File System groups these logical blocks into *allocation blocks*, which can contain one or more logical blocks, depending on the total size of the volume. HFS uses a 16-bit value to address allocation blocks, limiting the number of allocation blocks to 65,535 (2^{16}-1).

Five structures make up an HFS volume:

1. Logical blocks 0 and 1 of the volume are the **Boot Blocks**, which contain system startup information. For example, the names of the System and Shell (usually the Finder) files which are loaded at startup.

2. Logical block 2 contains the **Master Directory Block** (aka **MDB**). This defines a wide variety of data about the volume itself, for example date & time stamps for when the volume was created, the location of the other volume structures such as the Volume Bitmap or the size of logical structures such as allocation blocks. There is also a duplicate of the MDB called the **Alternate Master Directory Block** (aka **Alternate MDB**) located at the opposite end of the volume in the second to last logical block. This is intended mainly for use by disk utilities and is only updated when either the Catalog File or Extents Overflow File grow in size.
3. Logical block 3 is the starting block of the **Volume Bitmap**, which keeps track of which allocation blocks are in use and which are free. Each allocation block on the volume is represented by a bit in the map: if the bit is set then the block is in use; if it is clear then the block is free to be used. Since the Volume Bitmap must have a bit to represent each allocation block, its size is determined by the size of the volume itself.
4. The **Extent Overflow File** is a B-tree that contains extra extents that record which allocation blocks are allocated to which files, once the initial three extents in the Catalog File are used up. Later versions also added the ability for the Extent Overflow File to store extents that record bad blocks, to prevent the file system from trying to allocate a bad block to a file.
5. The **Catalog File** is another B-tree that contains records for all the files and directories stored in the volume. It stores four types of records. Each file consists of a File Thread Record and a File Record while each directory consists of a Directory Thread Record and a Directory Record. Files and directories in the Catalog File are located by their unique **Catalog Node ID** (or **CNID**).
    - A **File Thread Record** stores just the name of the file and the CNID of its parent directory.
    - A **File Record** stores a variety of metadata about the file including its CNID, the size of the file, three timestamps (when the file was created, last modified, last backed up), the first file extents of the data and resource forks and pointers to the file's first data and resource extent records in the Extent Overflow File. The File Record also stores two 16 byte fields that are used by the Finder to store attributes about the file including things like its creator code, type code, the window the file should appear in and its location within the window.
    - A **Directory Thread Record** stores just the name of the directory and the CNID of its parent directory.
    - A **Directory Record** which stores data like the number of files stored within the directory, the CNID of the directory, three timestamps (when the directory was created, last modified, last backed up). Like the File Record, the Directory Record also stores two 16 byte fields for use by the Finder. These store things like the width & height and x & y co-ordinates for the window used to display the contents of the directory, the display mode (icon view, list view, etc.) of the window and the position of the window's scroll bar.

Limitations

The Catalog File, which stores all the file and directory records in a single data structure, results in performance problems when the system allows multitasking, as only one program can write to this structure at a time, meaning that many programs may be waiting in queue due to one program "hogging" the system.[3] It is also a serious reliability concern, as damage to this file can destroy the entire file system. This contrasts with other file systems that store file and directory records in separate structures (such as DOS's FAT file system or the Unix File System), where having structure distributed across the disk means that damaging a single directory is generally non-fatal and the data may possibly be re-constructed with data held in the non-damaged portions.

Additionally, the limit of 65,535 allocation blocks resulted in files having a "minimum" size equivalent 1/65,535th the size of the disk. Thus, any given volume, no matter its size, could only store a maximum of 65,535 files. Moreover, any file would be allocated more space than it actually needed, up to the allocation

block size. When disks were small, this was of little consequence, because the individual allocation block size was trivial, but as disks started to approach the 1 GB mark, the smallest amount of space that any file could occupy (a single allocation block) became excessively large, wasting significant amounts of disk space. For example, on a 1 GB disk, the allocation block size under HFS is 16 KB, so even a 1 byte file would take up 16 KB of disk space. This situation was less of a problem for users having large files (such as pictures, databases or audio) because these larger files wasted less space as a percentage of their file size. Users with many small files, on the other hand, could lose a copious amount of space due to large allocation block size. This made partitioning disks into smaller logical volumes very appealing for Mac users, because small documents stored on a smaller volume would take up much less space than if they resided on a large partition. The same problem existed in the FAT16 file system.

HFS saves the case of a file that is created or renamed but is case-insensitive in operation.

According to bombich.com, HFS is no longer supported on Catalina and future macOS releases.

See also

- Comparison of file systems
- APFS
- HFS Plus

References

1. Gagne, Ken (2009-08-31). "Losing legacy data to Snow Leopard" (http://www.computerworld.com/article/2467506/mac-os-x/losing-legacy-data-to-snow-leopard.html). Computerworld. Retrieved 2009-09-07.
2. "What's New in macOS: macOS Sierra 10.12" (https://developer.apple.com/library/prerelease/content/releasenotes/MacOSX/WhatsNewInOSX/Articles/OSXv10.html#//apple_ref/doc/uid/TP40017145-SW1). Apple. Retrieved 25 January 2017.
3. Giampaolo, Dominic (1999). *Practical File System Design with the Be File System* (https://web.archive.org/web/20170213221835/http://www.nobius.org/~dbg/practical-file-system-design.pdf) (PDF). Morgan Kaufmann. p. 37. ISBN 1-55860-497-9. Archived from the original (http://www.nobius.org/~dbg/practical-file-system-design.pdf) (PDF) on 2017-02-13. Retrieved 2006-07-13.

External links

- HFS specification (https://developer.apple.com/legacy/mac/library/documentation/mac/Files/Files-99.html) from developer.apple.com
- The HFS Primer (http://www.macjournals.com/~mwj/mwj_samples/MWJ_20030525.pdf) (PDF) from MWJ (http://www.macjournals.com/mwj/) - dead link as of 27. May 2017
- Filesystems HOWTO: HFS (http://www.tldp.org/HOWTO/Filesystems-HOWTO-7.html#hfs) - slightly out of date
- HFS File Structure Explained (https://web.archive.org/web/20070927021136/http://mactech.com/articles/mactech/Vol.01/01.12/HFSFileStructure/) - early description of HFS
- DiskWarrior (https://web.archive.org/web/20060511103200/http://www.alsoft.com/DiskWarrior/) - Software to eliminate all damage to the HFS disk directory
- MacDrive (http://www.mediafour.com/products/macdrive/) - Software to read and write HFS/HFS Plus-formatted disks on Microsoft Windows
- hfsutils (http://www.mars.org/home/rob/proj/hfs/) - open-source software to manipulate HFS on Unix, DOS, Windows, OS/2