OpenHeFS: Hight-throughput extended File System

Amlal El Mahrouss

November 30, 2025

1 Overview

The High-throughput Extended File System (OpenHeFS) is a custom filesystem tailored for performance, structure, and compact representation. It uses red-black trees for directory indexing, sparse block slicing for file layout, and fixed-size metadata structures optimized for 512-byte sector alignment.

2 Constants and Macros

Name	Value / Description
`kOpenHeFSVersion`	0x0103
`kOpenHeFSMagic`	"OpenHeFS" (9-byte magic identifier)
`kOpenHeFSMagicLen`	9
`kOpenHeFSBlockLen`	512 bytes
`kOpenHeFSFileNameLen`	256 characters
`kOpenHeFSPartNameLen`	128 characters
`kOpenHeFSMinimumDiskSize`	128 GiB
`kOpenHeFSDefaultVolumeName`	"OpenHeFS Volume"
`kOpenHeFSINDStartOffset`	Offset after boot node
`kOpenHeFSSearchAllStr`	"*" (wildcard string)

3 Disk and File Metadata Enums

3.1 Drive Kind (`UInt8`)

- 0xC0: Hard Drive
- 0xC1: Solid State Drive
- 0x0C: Optical Drive
- 0xCC: USB Mass Storage
- 0xC4: SCSI Drive
- 0xC6: Flash Drive
- 0xFF: Unknown

3.2 Disk Status (`UInt8`)

- 0x18: Unlocked
- 0x19: Locked
- 0x1A: Error
- 0x1B: Invalid

3.3 Encoding Flags (`UInt16`)

- UTF-8
- UTF-16
- UTF-32
- UTF-16BE
- UTF-16LE
- UTF-32BE
- UTF-32LE
- UTF-8BE
- UTF-8LE
- Binary

3.4 File Kinds (`UInt16`)

- 0x00: Regular File
- 0x01: Directory
- 0x02: Block Device
- 0x03: Character Device
- 0x04: FIFO
- 0x05: Socket
- 0x06: Symbolic Link
- 0x07: Unknown

3.5 File Flags (`UInt32`)

- 0x000: None
- 0x100: ReadOnly
- 0x101: Hidden
- 0x102: System
- 0x103: Archive
- 0x104: Device

4 Structures

4.1 HEFS_BOOT_NODE

Acts as the superblock of the filesystem, provides necessary information about the disk and the filesystem itself.

- `fMagic`, `fVolName`, `fVersion`, `fChecksum`
- Sector and disk geometry: `fSectorCount`, `fSectorSize`, `fBadSectors`
- Drive info: `fDiskKind`, `fEncoding`, `fDiskStatus`, `fDiskFlags`, `fVID`
- Tree layout: `fStartIND`, `fEndIND`, `fINDCount`
- Reserved: `fStartIN`, `fEndIN`, `fStartBlock`, `fEndBlock`

4.2 HEFS_INDEX_NODE

Contains file metadata and block layout.

- `fHashPath`, `fFlags`, `fKind`, `fSize`, `fChecksum`
- `fSymLink` - if set, `fHashPath` represents the symlink target
- Time: `fCreated`, `fAccessed`, `fModified`, `fDeleted`
- Ownership: `fUID`, `fGID`, `fMode`
- Block data: `fOffsetSliceLow`, `fOffsetSliceHigh`

4.3 HEFS_INDEX_NODE_DIRECTORY

Red-black tree based directory node.

- `fHashPath`, `fFlags`, `fKind`, `fEntryCount`, `fChecksum`
- Time and ownership same as inode
- `fINSlices[kOpenHeFSSliceCount]` for storing child inode links
- RB-Tree Fields:
  - `fColor`: Red or Black
  - `fNext`/`fPrev`: Sibling pointers
  - `fChild`: Left or right child (implementation-defined)
  - `fParent`: Parent node

5 Timestamp Layout (ATime)

`ATime` is a 64-bit timestamp and allocation status tracker with the following structure:

- Bits 63-32: Year
- Bits 31-24: Month
- Bits 23-16: Day
- Bits 15-8: Hour

- Bits 7-0: Minute

Constants:

- `kOpenHeFSTimeInvalid = 0x0`
- `kOpenHeFSTimeMax = 0xFFFFFFFFFFFFFFFF - 1`

6 Filesystem API

Provided by `Kernel::OpenHeFS::HeFileSystemParser`.

- `Format(drive, flags, name)` - Format drive with OpenHeFS
- `CreateINodeDirectory(drive, flags, dir)`
- `RemoveINodeDirectory(drive, flags, dir)`
- `CreateINode(drive, flags, dir, name, kind)`
- `DeleteINode(drive, flags, dir, name, kind)`
- `INodeManip(drive, block, size, dir, kind, name, in)`

Internal helpers:

- `INodeCtlManip`, `INodeDirectoryCtlManip`

7 Conclusion

OpenHeFS provides a modern and compact approach to high-performance file storage. Its use of red-black trees, fixed-size metadata, slice-based sparse files, and minimal overhead makes it a strong candidate for performance-sensitive use cases.