Inodes in Linux File Systems

Understanding the Core of File Management

A Student's Guide to Theory and Practice

Command Line   File Systems   Data Structures

Date: June 18, 2025

What is a File System?

- Manages how data is stored and retrieved on a storage device (e.g., hard drive, SSD).
- Provides a hierarchical structure (directories and files).
- Responsible for organizing files, managing metadata, and controlling access.

Common File Systems

Linux: ext4, XFS, Btrfs

Windows: NTFS, FAT32

macOS: APFS, HFS+

Network: NFS, SMB

Introducing the Inode: The Unsung Hero

- **Inode** stands for **Index Node**.
- A fundamental data structure in Linux/Unix-like file systems.
- Stores metadata (information *about* a file or directory).

> **Crucial Point**
>
> An inode does **not** store the file's name or its actual data content directly.

Key Functions of an Inode

- Uniquely identifies a file or directory within its filesystem.
- Stores attributes: permissions, ownership, timestamps, size.

- Contains pointers to the data blocks where the file's content resides.
- Facilitates file operations: access, modification, deletion.

Anatomy of an Inode: Core Metadata Fields (ext4 example)

i_mode
File type (regular, directory, link, etc.) and permissions.

i_uid / i_gid
User ID (owner) and Group ID (group).

i_size_lo / i_size_high
File size in bytes (supports large files).

i_links_count
Number of hard links to this inode.

i_blocks_lo / i_blocks_hi
Number of disk blocks consumed by the file.

Inode Metadata: Ownership & Permissions (ext4 i_mode)

Ownership

i_uid

User ID of the file owner

i_gid

Group ID of the file owner

Permissions (i_mode)

Read (r), Write (w), Execute (x) for:

- Owner (User)
- Group
- Others

Special bits:

SUID, SGID, Sticky bit

Inode Metadata: Timestamps & Size (ext4 example)

Timestamps

(seconds since epoch, with extra fields for sub-second precision)

i_atime
Last **access** time

i_mtime
Last **modification** time (content change)

i_ctime
Last **inode change** time (metadata change)

i_crtime
File **creation** time (in newer ext4)

File Size

i_size_lo & i_size_high
Total size of the file in bytes

Why Two Fields?
Supports very large files by combining low and high 32-bit values

Inode Metadata: Data Block Pointers

Inodes store pointers to the actual data blocks on disk.

Direct Pointers

(e.g., 12 in classic ext inodes)

Point directly to data blocks. For small files.

Single Indirect

Points to a block that contains a list of direct block pointers.

Double Indirect

Points to a block of single indirect pointers.

Triple Indirect

Points to a block of double indirect pointers. For very large files.

> ### Modern Filesystems (ext4)
>
> Often use **Extents**: a contiguous range of blocks, more efficient for large files. (Indicated by `EXT4_EXTENTS_FL` in `i_flags`).

Ext4 Inode Structure: Notable Fields

i_flags

Controls file behavior (e.g., immutable, append-only, extents, inline data, encryption).

i_generation

File version number (useful for NFS).

i_file_acl_lo

Points to extended attribute block (e.g., for ACLs).

i_extra_isize

Size of extended inode fields beyond the original ext2 inode (default 256 bytes for ext4 inode, base was 128).

i_checksum_hi / l_i_checksum_lo

Inode checksum for integrity.

Inode Numbers: Unique Identifiers

- Each file and directory is assigned a unique **inode number** within its filesystem.
- The OS uses this number to locate the inode and its metadata.
- Inode 0 is undefined.

Special Inodes (ext4 example)

Inode 1 List of defective blocks	**Inode 3** User quota file
Inode 2 Root directory (/) of the filesystem	**Inode 8** Journal inode

Inode Allocation: When & How?

Fixed Allocation (e.g., ext4)

- A fixed number of inodes is allocated when the filesystem is created.
- Often based on a ratio (e.g., 1 inode per 16 KB of disk space).

> **Limitation:**
>
> This limits the maximum number of files, regardless of free disk space.

Dynamic Allocation (e.g., Btrfs)

- Inodes are allocated on demand as files are created.
- No pre-defined limit tied to filesystem creation parameters in the same way.

> **Note:**
>
> `df -i` often shows 0 total inodes for Btrfs as it's not a fixed pool.

Inode Tables: Organization (ext4 example)

- In ext4, inodes are stored in **inode tables**.
- Each **block group** in an ext4 filesystem has its own inode table.

Location Calculation

```
block_group = (inode_number - 1) / inodes_per_group
offset_in_table = (inode_number - 1) % inodes_per_group
```

- The inode table is a contiguous set of blocks.

Tracking Free Inodes: The Inode Bitmap (ext4 example)

- Filesystems like ext4 use an **inode bitmap** to keep track of free and used inodes.
- Each block group has its own inode bitmap.

Bitmap Structure

A bit in the bitmap corresponds to an inode in the inode table:

0
inode is free

1
inode is used

Efficiency

This allows for quick allocation of a new inode when a file is created.

How Inodes Work: File Creation

1. User creates a new file

   ```
   $ touch myfile.txt
   ```

2. Filesystem allocates a free inode from the inode bitmap.

3. Inode metadata is populated (owner, permissions, timestamps, etc.).

4. A directory entry is created:
   - Links the chosen filename (`myfile.txt`) to the allocated inode number.
   - This entry is stored in the data blocks of the parent directory.

5. If data is written, data blocks are allocated and pointers are stored in the inode.

How Inodes Work: File Access & Modification

1. User requests to open a file

```
$ cat myfile.txt
```

2. System traverses the directory structure to find `myfile.txt`.

3. The directory entry provides the inode number for `myfile.txt`.

4. The system reads the inode using the inode number.

5. Metadata (e.g., permissions) is checked.

6. If access is allowed, data block pointers in the inode are used to read/write file content.

7. Timestamps (`i_atime`, `i_mtime`, `i_ctime`) are updated as appropriate.

How Inodes Work: File Deletion & Inode Reclamation

1 User deletes a file

```
$ rm myfile.txt
```

2 ✕ The directory entry linking the filename to the inode number is removed.

3 ↓ The inode's **link count** (`i_links_count`) is decremented.

4 **If link count becomes zero:**

- The inode is marked as free in the inode bitmap.
- Data blocks pointed to by the inode are marked as free in the data block bitmap.
- The actual data is not immediately erased but is now available for overwriting.

Directory Structure: Linking Names to Inodes

A directory is a special type of file.

Its data blocks contain a list of **directory entries**.

Each directory entry consists of:

- A filename
- The inode number corresponding to that filename

Example: /home/user/file.txt

- Root directory (inode 2) contains entry for `home`
- `home` directory inode points to data blocks with entry for `user`
- `user` directory inode points to data blocks with entry for `file.txt`

Hard Links: Multiple Names, One Inode

A **hard link** creates an additional directory entry (filename) that points to an *existing* inode.

```
$ ln original_file.txt hard_link.txt
```

Characteristics

- Both `original_file.txt` and `hard_link.txt` share the **same inode number**.
- They point to the same metadata and the same data blocks.
- Modifying the file via any hard link affects all names.
- The inode's `i_links_count` increases for each hard link.
- File data is deleted only when `i_links_count` drops to 0.

✕ **Cannot span across different filesystems**

⚠ **Generally, cannot link to directories**

(to prevent loops and simplify `fsck`)

Soft (Symbolic) Links: A File Pointing to a Path

A **soft link** (or symbolic link) is a special file whose content is the *pathname* of another file or directory.

```
$ ln -s target_file.txt soft_link.txt
```

Characteristics

- `soft_link.txt` has its **own, separate inode number**.
- The inode for `soft_link.txt` stores the path to `target_file.txt`.
- If `target_file.txt` is deleted or moved, the soft link becomes "broken" or "dangling."

✓ Can span across different filesystems

✓ Can link to directories

Hard Links vs. Soft Links: Key Inode Differences

Feature	Hard Link	Soft Link (Symbolic Link)
Inode Number	Shares inode with target	Has its own, distinct inode
Data Reference	Points directly to data (via shared inode)	Stores the *pathname* of the target
Target Deletion	Link remains valid; data persists if other links exist	Link becomes broken/dangling
Cross Filesystem	✗ No	✓ Yes
Link to Directory	✗ Generally No (restricted)	✓ Yes
`ls -l` **Link Count**	Reflects shared inode's link count	Link count of soft link file itself (usually 1)

Viewing Inodes: ls -i and stat

ls -i

Displays the inode number as the first column for files/directories.

Basic usage:

```
$ ls -i myfile.txt
```

Long format with inode:

```
$ ls -li
```

stat

Displays detailed information about a file/directory, including its inode.

Detailed file information:

```
$ stat myfile.txt
```

Output includes: Inode number, File type, Permissions, UID, GID, Size, Links, Timestamps, Blocks.

Get only inode number:

```
$ stat -c '%i' myfile.txt
```

Checking Inode Usage: df -i

The `df -i` command displays inode usage statistics for mounted filesystems.

```
$ df -i
```

```
$ df -i /path/to/filesystem
```

Output Columns

Filesystem	Device name
Inodes	Total number of inodes in the filesystem
IUsed	Number of used inodes
IFree	Number of free inodes
IUse%	Percentage of inodes used

⚠ Critical Warning

Running out of inodes (`IFree` is 0 or very low) prevents new file creation, even if disk space is available.

Finding Files by Inode & Other find Uses

find /path -inum <inode_number>

Locates all files (hard links) associated with a specific inode number within `/path`.

Example:

```
$ find /home -inum 123456
```

Useful for finding all names of a hard-linked file or troubleshooting.

find /path -xdev -inum <inode_number>

Searches only within the filesystem of `/path` (does not cross mount points).

Example:

```
$ find /home -xdev -inum 123456
```

Dangerous but Useful

Delete files to free inodes:

```
$ find /path -type f -delete
```

Can be used to delete files, e.g., to free up inodes if many small, unnecessary files exist.

⚠ Use with extreme caution!

Inode Limitations & Advanced Concepts

⚠ Inode Exhaustion (Fixed Allocation FS)

Running out of inodes prevents new file creation. Critical for systems with many small files (e.g., mail servers, caches).

Inline Data (ext4 EXT4_INLINE_DATA_FL)

For very small files, data can be stored directly within the inode structure itself, avoiding data block allocation and improving access speed.

Btrfs Dynamic Inode Allocation

Btrfs allocates inodes dynamically, not from a fixed pool set at mkfs time. `df -i` shows 0 total inodes.

☆ Special Inodes

Reserved inode numbers for filesystem internal structures (e.g., root dir, journal, bad blocks list).

Inode Size (i_extra_isize)

ext4 default inode size is 256 bytes, extendable for more metadata.

Summary & Key Takeaways

- Inodes are central to Linux file management, storing metadata (not names or data).
- Each file/directory has a unique inode number per filesystem.
- Inode structure includes permissions, ownership, timestamps, size, and data block pointers.
- Hard links share an inode; soft links have their own inode pointing to a path.
- Commands like `ls -i`, `stat`, `df -i`, and `find -inum` are vital for inspection and troubleshooting.
- Understanding inode allocation (fixed vs. dynamic) and limitations is key for system administration.

Thank You!

Questions & Discussion