Index Nodes

In a regular UNIX filesystem, the inode stores all the metadata pertaining to the file (time stamps, block maps, extended attributes, etc), not the directory entry. To find the information associated with a file, one must traverse the directory files to find the directory entry associated with a file, then load the inode to find the metadata for that file. ext4 appears to cheat (for performance reasons) a little bit by storing a copy of the file type (normally stored in the inode) in the directory entry. (Compare all this to FAT, which stores all the file information directly in the directory entry, but does not support hard links and is in general more seek-happy than ext4 due to its simpler block allocator and extensive use of linked lists.)

The inode table is a linear array of `struct ext4_inode`. The table is sized to have enough blocks to store at least `sb.s_inode_size * sb.s_inodes_per_group` bytes. The number of the block group containing an inode can be calculated as `(inode_number - 1) / sb.s_inodes_per_group`, and the offset into the group's table is `(inode_number - 1) % sb.s_inodes_per_group`. There is no inode 0.

The inode checksum is calculated against the FS UUID, the inode number, and the inode structure itself.

The inode table entry is laid out in `struct ext4_inode`.

Offset	Size	Name	Description
0x0	__le16	i_mode	File mode. See the table i_mode below.
0x2	__le16	i_uid	Lower 16-bits of Owner UID.
0x4	__le32	i_size_lo	Lower 32-bits of size in bytes.
0x8	__le32	i_atime	Last access time, in seconds since the epoch. However, if the EA_INODE inode flag is set, this inode stores an extended attribute value and this field contains the checksum of the value.
0xC	__le32	i_ctime	Last inode change time, in seconds since the epoch. However, if the EA_INODE inode flag is set, this inode stores an extended attribute value and this field contains the lower 32 bits of the attribute value's reference count.
0x10	__le32	i_mtime	Last data modification time, in seconds since the epoch. However, if the EA_INODE inode flag is set, this inode stores an extended attribute value and this field contains the number of the inode that owns the extended attribute.
0x14	__le32	i_dtime	Deletion Time, in seconds since the epoch.
0x18	__le16	i_gid	Lower 16-bits of GID.
0x1A	__le16	i_links_count	Hard link count. Normally, ext4 does not permit an inode to have more than 65,000 hard links. This applies to files as well as directories, which means that there cannot be more than 64,998 subdirectories in a directory (each subdirectory's '..' entry counts as a hard link, as does the '.' entry in the directory itself). With the DIR_NLINK feature enabled, ext4 supports more than 64,998 subdirectories by setting this field to 1 to indicate that the number of hard links is not known.
0x1C	__le32	i_blocks_lo	Lower 32-bits of "block" count. If the huge_file feature flag is not set on the filesystem, the file consumes `i_blocks_lo` 512-byte blocks on disk. If huge_file is set and EXT4_HUGE_FILE_FL is NOT set in `inode.i_flags`, then the file consumes `i_blocks_lo + (i_blocks_hi << 32)` 512-byte blocks on disk. If huge_file is set and EXT4_HUGE_FILE_FL IS set in `inode.i_flags`, then this file consumes (`i_blocks_lo + i_blocks_hi` << 32) filesystem blocks on disk.
0x20	__le32	i_flags	Inode flags. See the table i_flags below.
0x24	4 bytes	i_osd1	See the table i_osd1 for more details.
0x28	60 bytes	i_block[EXT4_N_BLOCKS=15]	Block map or extent tree. See the section "The Contents of inode.i_block".
0x64	__le32	i_generation	File version (for NFS).
0x68	__le32	i_file_acl_lo	Lower 32-bits of extended attribute block. ACLs are of course one of many possible extended attributes; I think the name of this field is a result of the first use of extended attributes being for ACLs.
0x6C	__le32	i_size_high / i_dir_acl	Upper 32-bits of file/directory size. In ext2/3 this field was named i_dir_acl, though it was usually set to zero and never used.

Offset	Size	Name	Description
0x70	__le32	i_obso_faddr	(Obsolete) fragment address.
0x74	12 bytes	i_osd2	See the table i_osd2 for more details.
0x80	__le16	i_extra_isize	Size of this inode - 128. Alternately, the size of the extended inode fields beyond the original ext2 inode, including this field.
0x82	__le16	i_checksum_hi	Upper 16-bits of the inode checksum.
0x84	__le32	i_ctime_extra	Extra change time bits. This provides sub-second precision. See Inode Timestamps section.
0x88	__le32	i_mtime_extra	Extra modification time bits. This provides sub-second precision.
0x8C	__le32	i_atime_extra	Extra access time bits. This provides sub-second precision.
0x90	__le32	i_crtime	File creation time, in seconds since the epoch.
0x94	__le32	i_crtime_extra	Extra file creation time bits. This provides sub-second precision.
0x98	__le32	i_version_hi	Upper 32-bits for version number.
0x9C	__le32	i_projid	Project ID.

The `i_mode` value is a combination of the following flags:

Value	Description
0x1	S_IXOTH (Others may execute)
0x2	S_IWOTH (Others may write)
0x4	S_IROTH (Others may read)
0x8	S_IXGRP (Group members may execute)
0x10	S_IWGRP (Group members may write)
0x20	S_IRGRP (Group members may read)
0x40	S_IXUSR (Owner may execute)
0x80	S_IWUSR (Owner may write)
0x100	S_IRUSR (Owner may read)
0x200	S_ISVTX (Sticky bit)
0x400	S_ISGID (Set GID)
0x800	S_ISUID (Set UID)
	These are mutually-exclusive file types:
0x1000	S_IFIFO (FIFO)
0x2000	S_IFCHR (Character device)
0x4000	S_IFDIR (Directory)
0x6000	S_IFBLK (Block device)
0x8000	S_IFREG (Regular file)
0xA000	S_IFLNK (Symbolic link)
0xC000	S_IFSOCK (Socket)

The `i_flags` field is a combination of these values:

Value	Description
0x1	This file requires secure deletion (EXT4_SECRM_FL). (not implemented)
0x2	This file should be preserved, should undeletion be desired (EXT4_UNRM_FL). (not implemented)
0x4	File is compressed (EXT4_COMPR_FL). (not really implemented)
0x8	All writes to the file must be synchronous (EXT4_SYNC_FL).
0x10	File is immutable (EXT4_IMMUTABLE_FL).
0x20	File can only be appended (EXT4_APPEND_FL).
0x40	The dump(1) utility should not dump this file (EXT4_NODUMP_FL).
0x80	Do not update access time (EXT4_NOATIME_FL).
0x100	Dirty compressed file (EXT4_DIRTY_FL). (not used)
0x200	File has one or more compressed clusters (EXT4_COMPRBLK_FL). (not used)
0x400	Do not compress file (EXT4_NOCOMPR_FL). (not used)
0x800	Encrypted inode (EXT4_ENCRYPT_FL). This bit value previously was EXT4_ECOMPR_FL (compression error), which was never used.
0x1000	Directory has hashed indexes (EXT4_INDEX_FL).
0x2000	AFS magic directory (EXT4_IMAGIC_FL).
0x4000	File data must always be written through the journal (EXT4_JOURNAL_DATA_FL).
0x8000	File tail should not be merged (EXT4_NOTAIL_FL). (not used by ext4)
0x10000	All directory entry data should be written synchronously (see `dirsync`) (EXT4_DIRSYNC_FL).
0x20000	Top of directory hierarchy (EXT4_TOPDIR_FL).
0x40000	This is a huge file (EXT4_HUGE_FILE_FL).

Value	Description
0x80000	Inode uses extents (EXT4_EXTENTS_FL).
0x100000	Verity protected file (EXT4_VERITY_FL).
0x200000	Inode stores a large extended attribute value in its data blocks (EXT4_EA_INODE_FL).
0x400000	This file has blocks allocated past EOF (EXT4_EOFBLOCKS_FL). (deprecated)
0x01000000	Inode is a snapshot (`EXT4_SNAPFILE_FL`). (not in mainline)
0x04000000	Snapshot is being deleted (`EXT4_SNAPFILE_DELETED_FL`). (not in mainline)
0x08000000	Snapshot shrink has completed (`EXT4_SNAPFILE_SHRUNK_FL`). (not in mainline)
0x10000000	Inode has inline data (EXT4_INLINE_DATA_FL).
0x20000000	Create children with the same project ID (EXT4_PROJINHERIT_FL).
0x80000000	Reserved for ext4 library (EXT4_RESERVED_FL).
	Aggregate flags:
0x705BDFFF	User-visible flags.
0x604BC0FF	User-modifiable flags. Note that while EXT4_JOURNAL_DATA_FL and EXT4_EXTENTS_FL can be set with setattr, they are not in the kernel's EXT4_FL_USER_MODIFIABLE mask, since it needs to handle the setting of these flags in a special manner and they are masked out of the set of flags that are saved directly to i_flags.

The `osd1` field has multiple meanings depending on the creator:

Linux:

Offset	Size	Name	Description
0x0	__le32	l_i_version	Inode version. However, if the EA_INODE inode flag is set, this inode stores an extended attribute value and this field contains the upper 32 bits of the attribute value's reference count.

Hurd:

Offset	Size	Name	Description
0x0	__le32	h_i_translator	??

Masix:

Offset	Size	Name	Description
0x0	__le32	m_i_reserved	??

The `osd2` field has multiple meanings depending on the filesystem creator:

Linux:

Offset	Size	Name	Description
0x0	__le16	l_i_blocks_high	Upper 16-bits of the block count. Please see the note attached to i_blocks_lo.
0x2	__le16	l_i_file_acl_high	Upper 16-bits of the extended attribute block (historically, the file ACL location). See the Extended Attributes section below.
0x4	__le16	l_i_uid_high	Upper 16-bits of the Owner UID.
0x6	__le16	l_i_gid_high	Upper 16-bits of the GID.
0x8	__le16	l_i_checksum_lo	Lower 16-bits of the inode checksum.
0xA	__le16	l_i_reserved	Unused.

Hurd:

Offset	Size	Name	Description
0x0	__le16	h_i_reserved1	??
0x2	__u16	h_i_mode_high	Upper 16-bits of the file mode.
0x4	__le16	h_i_uid_high	Upper 16-bits of the Owner UID.
0x6	__le16	h_i_gid_high	Upper 16-bits of the GID.
0x8	__u32	h_i_author	Author code?

Masix:

Offset	Size	Name	Description
0x0	__le16	h_i_reserved1	??
0x2	__u16	m_i_file_acl_high	Upper 16-bits of the extended attribute block (historically, the file ACL location).
0x4	__u32	m_i_reserved2[2]	??

Inode Size

In ext2 and ext3, the inode structure size was fixed at 128 bytes (`EXT2_GOOD_OLD_INODE_SIZE`) and each inode had a disk record size of 128 bytes. Starting with ext4, it is possible to allocate a larger on-disk inode at format time for all inodes in the filesystem to provide space beyond the end of the original ext2 inode. The on-disk inode record size is recorded in the superblock as `s_inode_size`. The number of bytes actually used by struct ext4_inode beyond the original 128-byte ext2 inode is recorded in the `i_extra_isize` field for each inode, which allows struct ext4_inode to grow for a new kernel without having to upgrade all of the on-disk inodes. Access to fields beyond EXT2_GOOD_OLD_INODE_SIZE should be verified to be within `i_extra_isize`. By default, ext4 inode records are 256 bytes, and (as of August 2019) the inode structure is 160 bytes (`i_extra_isize = 32`). The extra space between the end of the inode structure and the end of the inode record can be used to store extended attributes. Each inode record can be as large as the filesystem block size, though this is not terribly efficient.

Finding an Inode

Each block group contains `sb->s_inodes_per_group` inodes. Because inode 0 is defined not to exist, this formula can be used to find the block group that an inode lives in: `bg = (inode_num - 1) / sb->s_inodes_per_group`. The particular inode can be found within the block group's inode table at `index = (inode_num - 1) % sb->s_inodes_per_group`. To get the byte address within the inode table, use `offset = index * sb->s_inode_size`.

Inode Timestamps

Four timestamps are recorded in the lower 128 bytes of the inode structure -- inode change time (ctime), access time (atime), data modification time (mtime), and deletion time (dtime). The four fields are 32-bit signed integers that represent seconds since the Unix epoch (1970-01-01 00:00:00 GMT), which means that the fields will overflow in January 2038. If the filesystem does not have orphan_file feature, inodes that are not linked from any directory but are still open (orphan inodes) have the dtime field overloaded for use with the orphan list. The superblock field `s_last_orphan` points to the first inode in the orphan list; dtime is then the number of the next orphaned inode, or zero if there are no more orphans.

If the inode structure size `sb->s_inode_size` is larger than 128 bytes and the `i_inode_extra` field is large enough to encompass the respective `i_[cma]time_extra` field, the ctime, atime, and mtime inode fields are widened to 64 bits. Within this “extra” 32-bit field, the lower two bits are used to extend the 32-bit seconds field to be 34 bit wide; the upper 30 bits are used to provide nanosecond timestamp accuracy. Therefore, timestamps should not overflow until May 2446. dtime was not widened. There is also a fifth timestamp to record inode creation time (crtime); this field is 64-bits wide and decoded in the same manner as 64-bit [cma]time. Neither crtime nor dtime are accessible through the regular stat() interface, though debugfs will report them.

We use the 32-bit signed time value plus (2^32 * (extra epoch bits)). In other words:

Extra epoch bits	MSB of 32-bit time	Adjustment for signed 32-bit to 64-bit tv_sec	Decoded 64-bit tv_sec	valid time range
0 0	1	0	`-0x80000000 - -0x00000001`	1901-12-13 to 1969-12-31
0 0	0	0	`0x000000000 - 0x07fffffff`	1970-01-01 to 2038-01-19
0 1	1	0x100000000	`0x080000000 - 0x0ffffffff`	2038-01-19 to 2106-02-07
0 1	0	0x100000000	`0x100000000 - 0x17fffffff`	2106-02-07 to 2174-02-25
1 0	1	0x200000000	`0x180000000 - 0x1ffffffff`	2174-02-25 to 2242-03-16
1 0	0	0x200000000	`0x200000000 - 0x27fffffff`	2242-03-16 to 2310-04-04
1 1	1	0x300000000	`0x280000000 - 0x2ffffffff`	2310-04-04 to 2378-04-22
1 1	0	0x300000000	`0x300000000 - 0x37fffffff`	2378-04-22 to 2446-05-10

This is a somewhat odd encoding since there are effectively seven times as many positive values as negative values. There have also been long-standing bugs decoding and encoding dates beyond 2038, which don't seem to be fixed as of kernel 3.12 and e2fsprogs 1.42.8. 64-bit kernels incorrectly use the extra epoch bits 1,1 for dates between 1901 and 1970. At some point the kernel will be fixed and e2fsck will fix this situation, assuming that it is run before 2310.