Ext2

From OSDev Wiki

The **Second Extended Filesystem** (**ext2fs**) is a rewrite of the original *Extended Filesystem* and as such, is also based around the concept of "inodes." Ext2 served as the de facto filesystem of Linux for nearly a decade from the early 1990s to the early 2000s when it was superseded by the journaling file systems ext3 and ReiserFS. It has native support for UNIX ownership / access rights, symbolic- and hard-links, and other properties that are common among UNIX-like operating systems. Organizationally, it divides disk space up into groups called "block groups." Having these groups results in distribution of data across the disk which helps to minimize head movement as well as the impact of fragmentation. Further, some (if not all) groups are required to contain backups of important data that can be used to rebuild the file system in the event of disaster.

Note: Most of the information here is based off of work done by Dave Poirier on the ext2-doc project (see the links section) which is graciously released under the GNU Free Documentation License (http://www.fsf.org/licenses/fdl.html) . Be sure to buy him a beer the next time you see him.

Filesystems

Virtual FileSystems

VFS

Disk filesystems

FAT 12/16/32, VFAT
Ext 2/3/4
LEAN
HPFS
NTFS
HFS
HFS+
MFS
ReiserFS
FFS (Amiga)
FFS (BSD)/UFS
BeFS
BFS
XFS
SFS
ZFS

CD/DVD filesystems

ISO 9660
Joliet
UDF

Network filesystems

NFS
RFS
AFS

Flash filesystems

JFFS2
YAFFS

Contents

Basic Concepts

Important Note: All values are little-endian unless otherwise specified

What is a Block?

The Ext2 file system divides up disk space into logical blocks of contiguous space. The size of blocks need not be the same size as the sector size of the disk the file system resides on. The size of blocks can be determined by reading the field starting at byte 24 in the Superblock.

What is a Block Group?

Blocks, along with inodes, are divvied up into "block groups." These are nothing more than contiguous groups of blocks.

Each block group reserves a few of its blocks for special purposes such as:

- A bitmap of free/allocated blocks within the group
- A bitmap of allocated inodes within the group
- A table of inode structures that belong to the group
- Depending upon the revision of Ext2 used, some or all block groups may also contain a backup copy of the Superblock and the Block Group Descriptor Table.

What is an Inode?

An inode is a structure on the disk that represents a file, directory, symbolic link, etc. Inodes do not contain the data of the file / directory / etc. that they represent. Instead, they link to the blocks that actually contain the data. This lets the inodes themselves have a well-defined size which lets them be placed in easily indexed arrays. Each block group has an array of inodes it is responsible for, and conversely every inode within a file system belongs to one of such tables (and one of such block groups).

Superblock

The first step in implementing an Ext2 driver is to find, extract, and parse the superblock. The Superblock contains all information about the layout of the file system and possibly contains other important information like what optional features were used to create the file system. Once you have finished with the Superblock, the next step is to look at the Block Group Descriptor Table

Locating the Superblock

The Superblock is always located at byte 1024 from the beginning of the volume and is exactly 1024 bytes in length. For example, if the disk uses 512 byte sectors, the Superblock will begin at LBA 2 and will occupy all of sector 2 and 3.

Determining the Number of Block Groups

From the Superblock, extract the size of each block, the total number of inodes, the total number of blocks, the number of blocks per block group, and the number of inodes in each block group. From this information we can infer the number of block groups there are by:

- Rounding up the total number of blocks divided by the number of blocks per block group
- Rounding up the total number of inodes divided by the number of inodes per block group
- Both (and check them against each other)

Base Superblock Fields

These fields are present in all versions of Ext2

Starting Byte	Ending Byte	Size in Bytes	Field Description
0	3	4	Total number of inodes in file system
4	7	4	Total number of blocks in file system
8	11	4	Number of blocks reserved for superuser (see offset 80)

12	15	4	Total number of unallocated blocks
16	19	4	Total number of unallocated inodes
20	23	4	Block number of the block containing the superblock
24	27	4	log_2 (block size) - 10. (In other words, the number to shift 1,024 to the left by to obtain the block size)
28	31	4	log_2 (fragment size) - 10. (In other words, the number to shift 1,024 to the left by to obtain the fragment size)
32	35	4	Number of blocks in each block group
36	39	4	Number of fragments in each block group
40	43	4	Number of inodes in each block group
44	47	4	Last mount time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
48	51	4	Last written time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
52	53	2	Number of times the volume has been mounted since its last consistency check (fsck (http://en.wikipedia.org/wiki/Fsck))
54	55	2	Number of mounts allowed before a consistency check (fsck (http://en.wikipedia.org/wiki/Fsck)) must be done
56	57	2	Ext2 signature (0xef53), used to help confirm the presence of Ext2 on a volume
58	59	2	File system state (see below)
60	61	2	What to do when an error is detected (see below)
62	63	2	Minor portion of version (combine with Major portion below to construct full version field)
64	67	4	POSIX time (http://en.wikipedia.org/wiki/Unix_time) of last consistency check (fsck (http://en.wikipedia.org/wiki/Fsck))
68	71	4	Interval (in POSIX time (http://en.wikipedia.org/wiki/Unix_time)) between forced consistency checks (fsck (http://en.wikipedia.org/wiki/Fsck))
72	75	4	Operating system ID from which the filesystem on this volume was created (see below)
76	79	4	Major portion of version (combine with Minor portion above to construct full version field)
80	81	2	User ID that can use reserved blocks
82	83	2	Group ID that can use reserved blocks

File System States

Value	State Description
1	File system is clean
2	File system has errors

Error Handling Methods

Value	Action to Take
1	Ignore the error (continue on)
2	Remount file system as read-only
3	Kernel panic

Creator Operating System IDs

Value	Operating System
0	Linux (http://kernel.org/)
1	GNU HURD (http://www.gnu.org/software/hurd/hurd.html)
2	MASIX (an operating system developed by Rémy Card, one of the developers of ext2)
3	FreeBSD (http://www.freebsd.org/)
4	Other "Lites" (BSD4.4-Lite derivatives such as NetBSD (http://www.netbsd.org/) , OpenBSD (http://www.openbsd.org/) , XNU/Darwin (http://www.opensource.apple.com/source/xnu/) , etc.)

Extended Superblock Fields

These fields are only present if Major version (specified in the base superblock fields), is greater than or equal to 1.

Starting Byte	Ending Byte	Size in Bytes	Field Description
84	87	4	First non-reserved inode in file system. (In versions < 1.0, this is fixed as 11)
88	89	2	Size of each inode structure in bytes. (In versions < 1.0, this is fixed as 128)
90	91	2	Block group that this superblock is part of (if backup copy)

92	95	4	Optional features present (features that are not required to read or write, but usually result in a performance increase. see below)
96	99	4	Required features present (features that are required to be supported to read or write. see below)
100	103	4	Features that if not supported, the volume must be mounted read-only see below)
104	119	16	File system ID (what is output by blkid)
120	135	16	Volume name (C-style string: characters terminated by a 0 byte)
136	199	64	Path volume was last mounted to (C-style string: characters terminated by a 0 byte)
200	203	4	Compression algorithms used (see Required features above)
204	204	1	Number of blocks to preallocate for files
205	205	1	Number of blocks to preallocate for directories
206	207	2	(Unused)
208	223	16	Journal ID (same style as the File system ID above)
224	227	4	Journal inode
228	231	4	Journal device
232	235	4	Head of orphan inode list
236	1023	X	(Unused)

Optional Feature Flags

These are optional features for an implementation to support, but offer performance or reliability gains to implementations that do support them.

Flag Value	Description
0x0001	Preallocate some number of (contiguous?) blocks (see byte 205 in the superblock) to a directory when creating a new one (to reduce fragmentation?)
0x0002	AFS server inodes exist
0x0004	File system has a journal (Ext3)
0x0008	Inodes have extended attributes
0x0010	File system can resize itself for larger partitions
0x0020	Directories use hash index

Required Feature Flags

These features if present on a file system are required to be supported by an implementation in order to correctly read from or write to the file system.

Flag Value	Description
0x0001	Compression is used
0x0002	Directory entries contain a type field
0x0004	File system needs to replay its journal
0x0008	File system uses a journal device

Read-Only Feature Flags

These features, if present on a file system, are required in order for an implementation to write to the file system, but are not required to read from the file system.

Flag Value	Description
0x0001	Sparse superblocks and group descriptor tables
0x0002	File system uses a 64-bit file size
0x0004	Directory contents are stored in the form of a Binary Tree (http://en.wikipedia.org/wiki/Binary_tree)

Block Group Descriptor Table

The Block Group Descriptor Table contains a descriptor for each block group within the file system. The number of block groups within the file system, and correspondingly, the number of entries in the Block Group Descriptor Table, is described above. Each descriptor contains information regarding where important data structures for that group are located.

Locating the Block Group Descriptor Table

The table is located in the block immediately following the Superblock. So if the block size (determined from a field in the superblock) is 1024 bytes per block, the Block Group Descriptor Table will begin at block 2. For any other block size, it will begin at block 1. Remember that blocks are numbered starting at 0, and that block numbers don't usually correspond to physical block addresses.

Block Group Descriptor

A Block Group Descriptor contains information regarding where important data structures for that block group are located.

Starting Byte	Ending Byte	Size in Bytes	Field Description
0	3	4	Block address of block usage bitmap
4	7	4	Block address of inode usage bitmap
8	11	4	Starting block address of inode table
12	13	2	Number of unallocated blocks in group
14	15	2	Number of unallocated inodes in group
16	17	2	Number of directories in group
18	31	X	(Unused)

Inodes

Like blocks, each inode has a numerical address. It is extremely important to note that unlike block addresses, **inode addresses start at 1**.

With Ext2 versions prior to Major version 1, inodes 1 to 10 are reserved and should be in an allocated state. Starting with version 1, the first non-reserved inode is indicated via a field in the Superblock. Of the reserved inodes, number 2 subjectively has the most significance as it is used for the root directory.

Inodes have a fixed size of either 128 for version 0 Ext2 file systems, or as dictated by the field in the Superblock for version 1 file systems. All inodes reside in inode tables that belong to block groups. Therefore, looking up an inode is simply a matter of determining which block group it belongs to and indexing that block group's inode table.

Determining which Block Group contains an Inode

From an inode address (remember that they start at 1), we can determine which group the inode is in, by using the formula:

```
block group = (inode – 1) / INODES_PER_GROUP
```

where INODES_PER_GROUP is a field in the Superblock

Finding an inode inside of a Block Group

Once we know which group an inode resides in, we can look up the actual inode by first retrieving that block group's inode table's starting address (see Block Group Descriptor above). The index of our inode in this block group's inode table can be determined by using the formula:

```
index = (inode – 1) % INODES_PER_GROUP
```

where % denotes the Modulo operation (http://en.wikipedia.org/wiki/Modulo_operation) and INODES_PER_GROUP is a field in the Superblock (the same field which was used to determine which block group the inode belongs to).

Next, we have to determine which block contains our inode. This is achieved from:

```
containing block = (index * INODE_SIZE) / BLOCK_SIZE
```

where INODE_SIZE is either fixed at 128 if VERSION < 1 or defined by a field in the Superblock if VERSION >= 1.0, and BLOCK_SIZE is defined by a field in the Superblock.

Finally, mask and shift as necessary to extract only the inode data from the containing block.

Reading the contents of an inode

Each inode contains 12 direct pointers, one singly indirect pointer, one doubly indirect block pointer, and one triply indirect pointer. The direct space "overflows" into the singly indirect space, which overflows into the doubly indirect space, which overflows into the triply indirect space.

Direct Block Pointers: There are 12 direct block pointers. If valid, the value is non-zero. Each pointer is the block address of a block containing data for this inode.

Singly Indirect Block Pointer: If a file needs more than 12 blocks, a separate block is allocated to store the block addresses of the remaining data blocks needed to store its contents. This separate block is called an indirect block because it adds an extra step (a level of indirection) between an inode and its data. The block addresses stored in the block are all 32-bit, and the capacity of stored addresses in this block is a function of the block size. The address of this indirect block is stored in the inode in the "Singly Indirect Block Pointer" field.

Doubly Indirect Block Pointer: If a file has more blocks than can fit in the 12 direct pointers and the indirect block, a double indirect block is used. A double indirect block is an extension of the indirect block described above only now we have two intermediate blocks between the inode and data blocks. The inode structure has a "Doubly Indirect Block Pointer" field that points to this block if necessary.

Triply Indirect Block Pointer: Lastly, if a file needs still more space, it can use a triple indirect block. Again, this is an extension of the double indirect block. So, a triple indirect block contains addresses of double indirect blocks, which contain addresses of single indirect blocks, which contain address of data blocks. The inode structure has a "Triply Indirect Block Pointer" field that points to this block if present.

This image from Wikipedia (http://en.wikipedia.org/wiki/File:Ext2-inode.gif) illustrates what is described above pretty well

Inode Data Structure

Starting Byte	Ending Byte	Size in Bytes	Field Description
0	1	2	Type and Permissions (see below)
2	3	2	User ID
4	7	4	Lower 32 bits of size in bytes
8	11	4	Last Access Time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
12	15	4	Creation Time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
16	19	4	Last Modification time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
20	23	4	Deletion time (in POSIX time (http://en.wikipedia.org/wiki/Unix_time))
24	25	2	Group ID
26	27	2	Count of hard links (directory entries) to this inode. When this reaches 0, the data blocks are marked as unallocated.
28	31	4	Count of disk sectors (not Ext2 blocks) in use by this inode, not counting the actual inode structure nor directory entries linking to the inode.
32	35	4	Flags (see below)
36	39	4	Operating System Specific value #1
40	43	4	Direct Block Pointer 0
44	47	4	Direct Block Pointer 1
48	51	4	Direct Block Pointer 2
52	55	4	Direct Block Pointer 3
56	59	4	Direct Block Pointer 4
60	63	4	Direct Block Pointer 5
64	67	4	Direct Block Pointer 6
68	71	4	Direct Block Pointer 7
72	75	4	Direct Block Pointer 8
76	79	4	Direct Block Pointer 9
80	83	4	Direct Block Pointer 10

84	87	4	Direct Block Pointer 11
88	91	4	Singly Indirect Block Pointer (Points to a block that is a list of block pointers to data)
92	95	4	Doubly Indirect Block Pointer (Points to a block that is a list of block pointers to Singly Indirect Blocks)
96	99	4	Triply Indirect Block Pointer (Points to a block that is a list of block pointers to Doubly Indirect Blocks)
100	103	4	Generation number (Primarily used for NFS)
104	107	4	In Ext2 version 0, this field is reserved. In version >= 1, Extended attribute block (File ACL).
108	111	4	In Ext2 version 0, this field is reserved. In version >= 1, Upper 32 bits of file size (if feature bit set) if it's a file, Directory ACL if it's a directory
112	115	4	Block address of fragment
116	127	12	Operating System Specific Value #2

Inode Type and Permissions

The type indicator occupies the top hex digit (bits 15 to 12) of this 16-bit field

Type value in hex	Type Description
0x1000	FIFO
0x2000	Character device
0x4000	Directory
0x6000	Block device
0x8000	Regular file
0xA000	Symbolic link
0xC000	Unix socket

Permissions occupy the bottom 12 bits of this 16-bit field

Permission value in hex	Permission value in octal	Permission Description

0x001	00001	Other—execute permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x002	00002	Other—write permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x004	00004	Other—read permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x008	00010	Group—execute permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x010	00020	Group—write permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x020	00040	Group—read permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x040	00100	User—execute permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x080	00200	User—write permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x100	00400	User—read permission (http://en.wikipedia.org /wiki/Filesystem_permissions#Traditional_Unix_permissions)
0x200	01000	Sticky Bit (http://en.wikipedia.org/wiki/Sticky_bit)
0x400	02000	Set group ID
0x800	04000	Set user ID

Inode Flags

Flag Value	Description
0x00000001	Secure deletion (not used)
0x00000002	Keep a copy of data when deleted (not used)
0x00000004	File compression (not used)
0x00000008	Synchronous updates—new data is written immediately to disk
0x00000010	Immutable file (content cannot be changed)
0x00000020	Append only
0x00000040	File is not included in 'dump' command
0x00000080	Last accessed time should not updated
...	(Reserved)
0x00010000	Hash indexed directory
0x00020000	AFS directory

0x00040000	Journal file data

OS Specific Value 1

Operating System	How they use this field
Linux	(reserved)
HURD	"translator"?
MASIX	(reserved)

OS Specific Value 2

Operating System	How they use this field			
	Starting Byte	**Ending Byte**	**Size in Bytes**	**Field Description**
Linux	116	116	1	Fragment number
	117	117	1	Fragment size
	118	119	2	(reserved)
	120	121	2	High 16 bits of 32-bit User ID
	122	123	2	High 16 bits of 32-bit Group ID
	124	127	4	(reserved)
	Starting Byte	**Ending Byte**	**Size in Bytes**	**Field Description**
HURD	116	116	1	Fragment number
	117	117	1	Fragment size
	118	119	2	High 16 bits of 32-bit "Type and Permissions" field

	120	121	2	High 16 bits of 32-bit User ID
	122	123	2	High 16 bits of 32-bit Group ID
	124	127	4	User ID of author (if == 0xFFFFFFFF, the normal User ID will be used)
MASIX	**Starting Byte**	**Ending Byte**	**Size in Bytes**	**Field Description**
	116	116	1	Fragment number
	117	117	1	Fragment size
	118	127	X	(reserved)

Directories

Directories are inodes which contain some number of "entries" as their contents. These entries are nothing more than a name/inode pair. For instance the inode corresponding to the root directory might have an entry with the name of "etc" and an inode value of 50. A directory inode stores these entries in a linked-list fashion in its contents blocks.

The root directory is Inode 2.

Directory Entry

Starting Byte	Ending Byte	Size in Bytes	Field Description
0	3	4	Inode
4	5	2	Total size of this entry (Including all subfields)
6	6	1	Name Length least-significant 8 bits
7	7	1	Type indicator (only if the feature bit for "directory entries have file type byte" is set, else this is the most-significant 8 bits of the Name Length)
8	8+N-1	N	Name characters

Directory Entry Type Indicators

Value	Type Description
0	Unknown type
1	Regular file
2	Directory
3	Character device
4	Block device
5	FIFO
6	Socket
7	Symbolic link (soft link)

Quick Summaries

How To Read An Inode

1. Read the Superblock to find the size of each block, the number of blocks per group, number Inodes per group, and the starting block of the first group (Block Group Descriptor Table).
2. Determine which block group the inode belongs to.
3. Read the Block Group Descriptor corresponding to the Block Group which contains the inode to be looked up.
4. From the Block Group Descriptor, extract the location of the block group's inode table.
5. Determine the index of the inode in the inode table.
6. Index the inode table (taking into account non-standard inode size).

Directory entry information and file contents are located within the data blocks that the Inode points to.

How To Read the Root Directory

The root directory's inode is defined to always be 2. Read/parse the contents of inode 2.

See Also

External Links

- ext2-doc project: Second Extended File System (http://www.nongnu.org/ext2-doc/) - implementation-oriented documentation, describes internal structure in human language.
- Design and Implementation of the Second Extended Filesystem (http://web.mit.edu /tytso/www/linux/ext2intro.html) (overview)
- State of the Art: Where we are with the Ext3 filesystem (http://ext2.sourceforge.net

/2005-ols/paper-html/) - Paper by Mingming Cao, Theodore Y. Ts'o, Badari Pulavarty, and Suparna Bhattacharya describing extended features for ext2