Extended Attributes

Extended attributes (xattrs) are typically stored in a separate data block on the disk and referenced from inodes via `inode.i_file_acl*`. The first use of extended attributes seems to have been for storing file ACLs and other security data (selinux). With the `user_xattr` mount option it is possible for users to store extended attributes so long as all attribute names begin with "user"; this restriction seems to have disappeared as of Linux 3.0.

There are two places where extended attributes can be found. The first place is between the end of each inode entry and the beginning of the next inode entry. For example, if inode.i_extra_isize = 28 and sb.inode_size = 256, then there are 256 - (128 + 28) = 100 bytes available for in-inode extended attribute storage. The second place where extended attributes can be found is in the block pointed to by `inode.i_file_acl`. As of Linux 3.11, it is not possible for this block to contain a pointer to a second extended attribute block (or even the remaining blocks of a cluster). In theory it is possible for each attribute's value to be stored in a separate data block, though as of Linux 3.11 the code does not permit this.

Keys are generally assumed to be ASCIIZ strings, whereas values can be strings or binary data.

Extended attributes, when stored after the inode, have a header `ext4_xattr_ibody_header` that is 4 bytes long:

Offset	Type	Name	Description
0x0	__le32	h_magic	Magic number for identification, 0xEA020000. This value is set by the Linux driver, though e2fsprogs doesn't seem to check it(?)

The beginning of an extended attribute block is in `struct ext4_xattr_header`, which is 32 bytes long:

Offset	Type	Name	Description
0x0	__le32	h_magic	Magic number for identification, 0xEA020000.
0x4	__le32	h_refcount	Reference count.
0x8	__le32	h_blocks	Number of disk blocks used.
0xC	__le32	h_hash	Hash value of all attributes.
0x10	__le32	h_checksum	Checksum of the extended attribute block.
0x14	__u32	h_reserved[2]	Zero.

The checksum is calculated against the FS UUID, the 64-bit block number of the extended attribute block, and the entire block (header + entries).

Following the `struct ext4_xattr_header` or `struct ext4_xattr_ibody_header` is an array of `struct ext4_xattr_entry`; each of these entries is at least 16 bytes long. When stored in an external block, the `struct ext4_xattr_entry` entries must be stored in sorted order. The sort order is `e_name_index`, then `e_name_len`, and finally `e_name`. Attributes stored inside an inode do not need be stored in sorted order.

Offset	Type	Name	Description
0x0	__u8	e_name_len	Length of name.
0x1	__u8	e_name_index	Attribute name index. There is a discussion of this below.
0x2	__le16	e_value_offs	Location of this attribute's value on the disk block where it is stored. Multiple attributes can share the same value. For an inode attribute this value is relative to the start of the first entry; for a block this value is relative to the start of the block (i.e. the header).
0x4	__le32	e_value_inum	The inode where the value is stored. Zero indicates the value is in the same block as this entry. This field is only used if the INCOMPAT_EA_INODE feature is enabled.
0x8	__le32	e_value_size	Length of attribute value.
0xC	__le32	e_hash	Hash value of attribute name and attribute value. The kernel doesn't update the hash for in-inode attributes, so for that case this value must be zero, because e2fsck validates any non-zero hash regardless of where the xattr lives.
0x10	char	e_name[e_name_len]	Attribute name. Does not include trailing NULL.

Attribute values can follow the end of the entry table. There appears to be a requirement that they be aligned to 4-byte boundaries. The values are stored starting at the end of the block and grow towards the xattr_header/xattr_entry table. When the two collide, the overflow is put into a separate disk block. If the disk block fills up, the filesystem returns -ENOSPC.

The first four fields of the `ext4_xattr_entry` are set to zero to mark the end of the key list.

Attribute Name Indices

Logically speaking, extended attributes are a series of key=value pairs. The keys are assumed to be NULL-terminated strings. To reduce the amount of on-disk space that the keys consume, the beginning of the key string is matched against the attribute name index. If a match is found, the attribute name index field is set, and matching string is removed from the key name. Here is a map of

name index values to key prefixes:

Name Index	Key Prefix
0	(no prefix)
1	“user.”
2	“system.posix_acl_access”
3	“system.posix_acl_default”
4	“trusted.”
6	“security.”
7	“system.” (inline_data only?)
8	“system.richacl” (SuSE kernels only?)

For example, if the attribute key is “user.fubar”, the attribute name index is set to 1 and the “fubar” name is recorded on disk.

POSIX ACLs

POSIX ACLs are stored in a reduced version of the Linux kernel (and libacl's) internal ACL format. The key difference is that the version number is different (1) and the `e_id` field is only stored for named user and group ACLs.