Betriebssysteme I – Sommersemester 2009

Kapitel 6: Speicherverwaltung und Dateisysteme

Hans-Georg Eßer
Hochschule München

Teil 4: Exkurs: Linux-Dateisystem

06/2009

Gliederung

- Virtual Filesystem (VFS) allgemein
- Linux VFS
- Unix-Dateiattribute
- Ext2/Ext3-Dateisystem

Virtuelles Dateisystem – VFS (1)

Zwei Schichten einführen

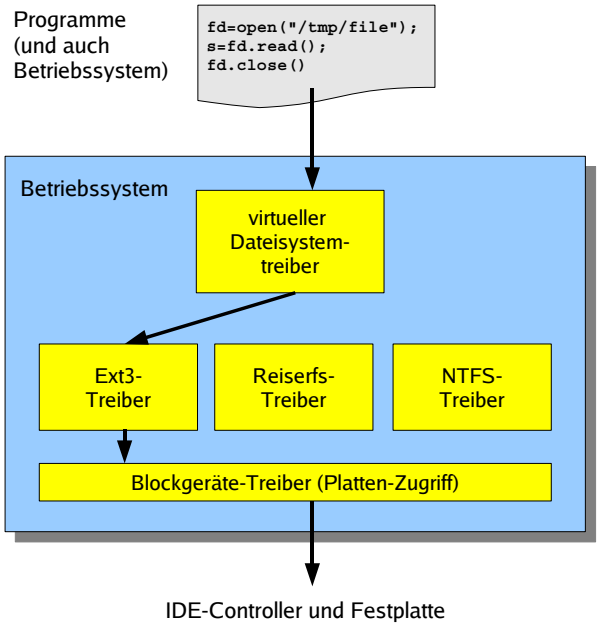
- VFS-Treiber stellt High-Level-Dateioperationen bereit (create, delete, rename, open, close, read, write, seek, link, ...)
- Kommunikation aller Programme (und auch des BS selbst) nur mit dem VFS-Treiber
- VFS-Treiber leitet Anfragen an Spezialtreiber für die Dateisysteme weiter
- Spezialtreiber beherrschen einzelne FS

VFS (2)

VFS-Treiber
- kennt abstrakte Datei-eigenschaften
- weiß, welchen speziellen Dateisystem-Treiber er braucht

Dateisystem-Treiber
- kennt das jeweilige Dateisystem-Format (weiß nichts von HW)

Blockgeräte-Treiber
- kann mit der Hardware sprechen



VFS (3): Standard-Funktionen

- In jedem Betriebssystem unterstützt das VFS mindestens
  - create (); Datei erzeugen
  - delete (); Datei löschen
  - open (); Datei öffnen
  - close (); offene Datei schließen
  - read (); aus Datei lesen
  - write (); in Datei schreiben
  - append (); ans Ende der Datei etwas anhängen

VFS (4): Standard-Funktionen

- seek (); in Datei an bestimmte Stelle springen
- get_attrib (); Datei-Eigenschaften abfragen (was auch immer das BS hier für Eigenschaften zulässt)
- set_attrib (); Datei-Eigenschaften setzen
- rename (); Dateinamen ändern

- Das VFS kennt die Datei-Attribute und -Operationen, die für das Betriebssystem relevant sind

Linux VFS (1)

Vier elementare Konzepte

- **Datei:** Folge von Bytes, keine „Dateitypen“ (aus Sicht des Dateisystems)
- **Verzeichnis:** Spezialdatei mit Informationen über Dateien im Verzeichnis
- **Inode:** Index node, enthält die Datei-Metadaten
- **Mount-Punkt:** Einbinden eines Dateisystems in den Verzeichnisbaum; der Mount-Punkt ist die Wurzel des Dateisystems

Linux VFS (2)

Inodes

- zentrale Einheit in Linux/Unix ist der **Inode** (Information Node)
- Dateisystem verwaltet eine Inode-Liste; jede Datei verwendet einen Inode
- Zeiger auf die eigentlichen Daten
- Datei erzeugen =
  - Platz reservieren,
  - freien Inode suchen
  - Verwaltungsinformationen in den Inode schreiben

Linux VFS (3)

Inode-**Metadaten**:

- Inode-Nummer
- Anzahl der **Links** (Einträge dieses Inodes in Verzeichnissen)
- User-ID (Besitzer), Group-ID (Besitzergruppe)
- Dateigröße
- Zugriffszeiten:
  - ctime (creation time)
  - mtime (last modification time)
  - atime (last access time)
- **Vorsicht: kein Dateiname**

Linux VFS (4)

Dateien

- File-Objekt: geöffnete Datei
- Prozesse arbeiten mit File-Objekten, Ansprache über **File Descriptor** (fd)
- Objekt wird bei open()-Aufruf erzeugt und bei close()-Aufruf zerstört
- Es kann mehrere File-Objekte zur gleichen Datei geben (gemeinsamer Zugriff)

Linux VFS (5)

Verzeichnisse

- Auch ein Verzeichnis ist eine Datei (und zwar eine, die Hinweise auf den Ort weiterer Dateien enthält)
- Verzeichniseintrag (directory entry): Name + Inode-Nummer
- Eine Datei kann in mehreren Verzeichnissen (oder mehrfach im gleichen Verzeichnis) auftreten (➞ Hard Links)

Linux VFS Standard-Funktionen (1)

- **fd = open (filename, flags)**
  Datei zum Lesen, Schreiben öffnen; auch: erzeugen
- **close (fd)** offene Datei schließen
- **link ()** erzeugt neuen Verweis auf eine Datei
- **lseek (fd, offset, Art)** springt an eine andere Stelle in der Datei (abhängig vom letzten Parameter absolut, relativ oder hinter dem Dateiende)
- **read (fd, buffer, count), write (fd, buffer, count)**
- **stat (filename, status), fstat (fd, status)**
  Informationen über Datei abrufen
- **unlink (name)** Eintrag in einem Verzeichnis löschen – evtl. auch die Datei, wenn dies der letzte Link war

Linux VFS
Standard-Funktionen (2)

- **rename (oldpath, newpath)**
  Dateinamen (in einem Verzeichnis) ändern
  (kann auch in ein anderes Verzeichnis verschieben)
- **mmap (start, laenge, ..., fd, offset)**
  Datei ab Position *offset* mit Länge *laenge* in den Hauptspeicher einblenden
- **mkdir (pathname, mode)** Verzeichnis erzeugen
- **rmdir (pathname)** leeres Verzeichnis löschen
- **chdir (pathname)** Arbeitsverzeichnis wechseln
- **getcwd (*buf, size)**
  Name des Arbeitsverzeichnis in Puffer schreiben

Unix-Dateiattribute (1)

- Besitzer (**u**ser) und Gruppe (**g**roup)
- Lese- (**r**), Schreib- (**w**) und Ausführrechte (**x**) für Besitzer (**u**), Gruppe (**g**) und sonstige Benutzer (**o**, others)
- ergibt 9 Zugriffsrechte; typische Notation:

```
-rwxrwxrwx   Besitzer
             Gruppe
             sonstige
```

- Anwender können zu verschiedenen Gruppen gehören

Unix-Dateiattribute (2)

- numerische Rechte:
  - Leserecht: 4 (2^2)
  - Schreibrecht: 2 (2^1)
  - Ausführrecht: 1 (2^0)
  - aufaddieren, z. B.: Lesen/Schreiben: 4+2=6
- für Benutzer, Gruppe und Sonstige: **nnn**
  - z. B. **640**:
    - Benutzer: 6 = lesen + schreiben (nicht ausführen)
    - Gruppe: 4 = lesen (nicht schreiben, nicht ausführen)
    - Sonstige: 0 = nichts

Unix-Dateiattribute (3)

- Beim Erzeugen einer Datei werden Standardrechte gesetzt – welche das sind, bestimmt die UMASK (**u**ser file creation **mask**)

```
$ umask a=rw
$ umask
0111
$ touch Datei; ls -l Datei
-rw-rw-rw-  1 esser users 0 2008-12-04 20:48 Datei
$ umask u=rw,g=r,o=
$ umask
0137
$ touch Test; ls -l Test
-rw-r-----  1 esser users 0 2008-12-04 20:50 Test
```

Unix-Dateiattribute (4)

- umask wird von 777 (Maximalrechte) bitweise abgezogen, um konkrete Dateirechte zu berechnen;
- Ausführrecht wird beim Erzeugen einer Datei nie vergeben
- Linux unterstützt diese klassischen Unix-Dateiattribute und einige zusätzliche...

Unix-Dateiattribute (5)

- Dateiattribute nur auf echten Unix-Dateisystemen nutzbar – nicht auf Windows-Datenträgern:

```
# mount | grep windows
/dev/sda3 on /windows/D type vfat (rw,gid=100,umask=0002)
# touch /windows/D/Testdatei
# ls -l /windows/D/Testdatei
-rwxrwxr-x  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
# chmod a-rwx /windows/D/Testdatei
# ls -l /windows/D/Testdatei
----------  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
# umount /windows/D; mount /windows/D; ls -l /windows/D/Testdatei
-r-xr-xr-x  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
```

- Windows kennt kein Ausführattribut – wohl aber ein Read-Only-Attribut

Ext2-/Ext3-Dateisystem

- Ext3 ist heute das Standard-Linux-Dateisystem (neben ReiserFS)
- Geschichte
  - am Anfang war Minix...
  - Extended Filesystem als Ersatz für Minix entwickelt
  - Ext2 (2nd Extended Filesystem), Nachfolger von Ext
  - Ext3 (3rd Extended Filesystem) erweitert Ext2 um Journaling
  - Ganz neu: Ext4 (4th Extended Filesystem) mit „Extents“ (bis zu vier große zus.-hgd. Bereiche pro Datei)

Ext2/Ext3: Features (1)

- Features
  - variable Blockgrößen (kann je nach Einstellung gut mit vielen kleinen oder wenigen großen Dateien umgehen)
  - Schnelle symbolische Links (Link-Ziel direkt im Inode speichern, wenn die Adresse kurz ist)
  - Spezialattribute (die über die üblichen Unix-Dateiattribute hinaus gehen), z. B. *immutable*-Flag
  - Extended Attributes, z. B. ACLs
  - Kompatibilitäts-Bitmap für Weiterentwicklungen (*read/write*-, *read-only*- und *incompatible*-Bits)

Ext2/Ext3: Features (2)

- Features
  - stellt regelmäßige Überprüfung des Dateisystems sicher, auf zweierlei Basis:
    - Mount-Count: nur n-mal ohne Check mounten, dann prüfen
    - Zeit seit der letzten Überprüfung (maximales Intervall)
  - sicheres Löschen (durch Überschreiben der Daten)
  - nur Ext3: Journaling
    - macht im Normalbetrieb langwierige fsck-Läufe überflüssig
    - beschleunigt im Fehlerfall die Überprüfung mit fsck

Ext2/Ext3: Aufbau (1)

- Aufbau: Partition in Boot-Sektor und **Blockgruppen** unterteilt
- Jede Blockgruppe enthält eine Kopie des **Superblocks** (mit Verwaltungsinformationen des ganzen Dateisystems)

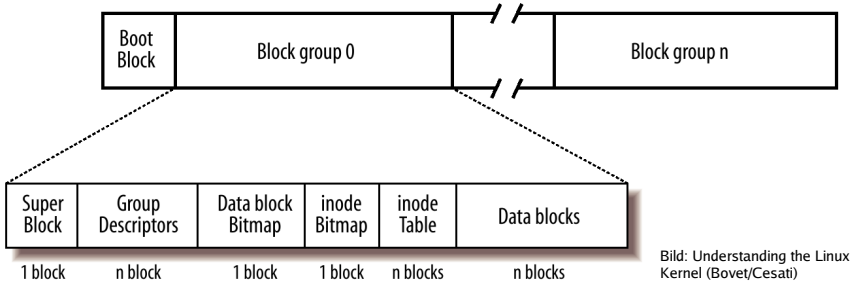
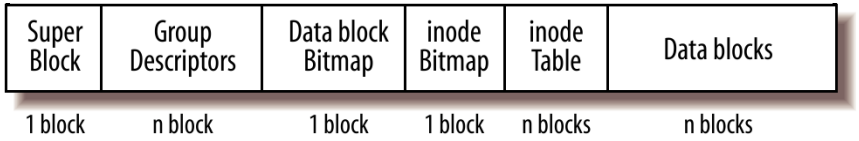


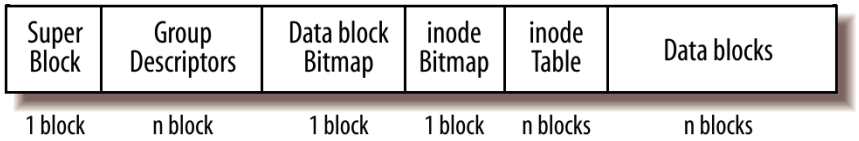
Bild: Understanding the Linux Kernel (Bovet/Cesati)

Ext2/Ext3: Aufbau (2)



- **Gruppen-Deskriptor:** Zustand der Blockgruppe (u.a. freie Blöcke und freie Inodes) (variable Länge)
  - Jede Blockgruppe enthält Deskriptoren für *alle* Blockgruppen
- **Datenblock-Bitmap:** Für jeden Datenblock ein Bit (frei/nicht frei)

Ext2/Ext3: Aufbau (3)



- **Inode-Bitmap:** Für jeden Inode ein Bit: benutzt/nicht benutzt
- **Inode-Tabelle:** Hier liegen alle Inodes der Blockgruppe (variable Länge)
- **Datenblöcke:** die eigentlichen Nutzdaten, also Dateien und Verzeichnisse

Ext2/Ext3: Aufbau (4)

- Metainformationen in jeder Blockgruppe – Idee:
  - Zerstört ein Absturz den Superblock, gibt es noch redundante Kopien
  - kleine Distanz zwischen Metainformationen und Daten: kürzere Seek-Zeiten beim Plattenzugriff
- Tatsächliche Umsetzung anders:
  - Kernel hält im RAM Kopie des Superblocks, den es nur bei *fsck*-Aufrufen auf die Superblöcke verteilt
  - Spätere Ext2-Versionen: *Sparse-Superblock*-Option. Superblöcke nur in Gruppen 0, 1, 3^n, 5^n, 7^n (n>1)

Indirektion bei Ext2 (1/2)

- 12 direkte Zeiger
- 1 einfach indirekter Zeiger
- 1 zweifach indirekter Zeiger
- 1 dreifach indirekter Zeiger

Block-größe	nur direkte Zeiger	maximal 1-fache Indir.	maximal 2-fache Indir.	3-fache Indirektion
1.024	12 KByte	268 KByte	64,26 MByte	16,06 GByte
2.048	24 KByte	1,02 MByte	513,02 MByte	256,50 GByte
4.096	48 KByte	4,04 MByte	4 GByte	~ 4 TByte

Indirektion bei Ext2 (2/2)

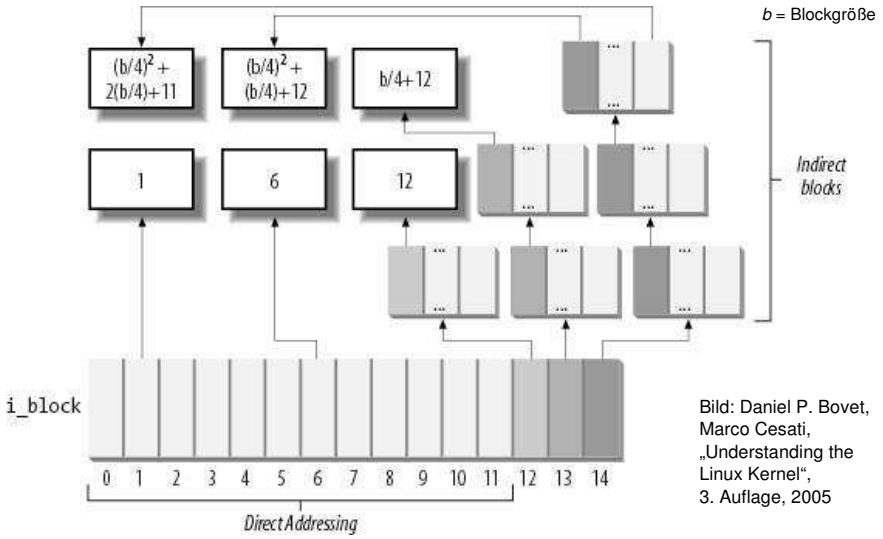


Bild: Daniel P. Bovet, Marco Cesati, „Understanding the Linux Kernel", 3. Auflage, 2005

Ext2-/Ext3-Superblock (1)

Datei: *include/linux/ext2_fs.h*

```
struct ext2_super_block {
    __le32  s_inodes_count;      /* Inodes count */
    __le32  s_blocks_count;      /* Blocks count */
```

Wie viele Inodes und Blöcke hat das Dateisystem?

```
    __le32  s_r_blocks_count;    /* Reserved blocks count */
    __le32  s_free_blocks_count; /* Free blocks count */
    __le32  s_free_inodes_count; /* Free inodes count */
```

Freie Inodes und Blöcke

```
    __le32  s_first_data_block;  /* First Data Block */
    __le32  s_log_block_size;    /* Block size */
```

Logarithmische Blockgröße:
1024 * 2^i, i=0,1,2 (1024, 2048, 4096)

```
    __le32  s_log_frag_size;     /* Fragment size */
    __le32  s_blocks_per_group;  /* # Blocks per group */
    __le32  s_frags_per_group;   /* # Fragments per group */
    __le32  s_inodes_per_group;  /* # Inodes per group */
```

Inodes / Blöcke in einer Blockgruppe

Ext2-/Ext3-Superblock (2)

```
__le32  s_mtime;            /* Mount time */
__le32  s_wtime;            /* Write time */
__le16  s_mnt_count;        /* Mount count */
__le16  s_max_mnt_count;    /* Maximal mount count */
```

Mount-Count wird bei jedem Einbinden hochgezählt; wird ein Maximalwert erreicht, ist ein *fsck* fällig

```
/* Maximal mount counts between two filesystem checks */
#define EXT2_DFL_MAX_MNT_COUNT      20  /* Allow 20 mounts */

__le16  s_magic;            /* Magic signature */

/* The second extended file system magic number */
#define EXT2_SUPER_MAGIC  0xEF53

__le16  s_state;            /* File system state */

/*
 * File system states
 */
#define  EXT2_VALID_FS          0x0001   /* Unmounted cleanly */
#define  EXT2_ERROR_FS          0x0002   /* Errors detected */
```


Ext2-/Ext3-Superblock (3)

```
__le16  s_errors;           /* Behaviour when detecting errors */

/*Behaviour when detecting errors */
#define EXT2_ERRORS_CONTINUE    1   /* Continue execution */
#define EXT2_ERRORS_RO          2   /* Remount fs read-only */
#define EXT2_ERRORS_PANIC       3   /* Panic */
#define EXT2_ERRORS_DEFAULT     EXT2_ERRORS_CONTINUE

__le16  s_minor_rev_level;  /* minor revision level */
__le32  s_lastcheck;        /* time of last check */
__le32  s_checkinterval;    /* max. time between checks */
```

lastcheck und checkinterval: Neben dem Mount-Count wird auch nach Ablauf einer bestimmten Zeit ein *fsck* durchgeführt

```
__le32  s_creator_os;       /* OS */
__le32  s_rev_level;        /* Revision level */
__le16  s_def_resuid;       /* Default uid for reserved blocks */
__le16  s_def_resgid;       /* Default gid for reserved blocks */
```

„Reserve“ (5 %) für privilegierten Benutzer.
Normal: resuid = resgid = 0 (root/root)

```
#define  EXT2_DEF_RESUID     0
#define  EXT2_DEF_RESGID     0
```


Ext2-/Ext3-Superblock (4)

```
__le32  s_first_ino;        /* First non-reserved inode */
__le16  s_inode_size;       /* size of inode structure */
__le16  s_block_group_nr;   /* block group # of this superblock */
__le32  s_feature_compat;   /* compatible feature set */
__le32  s_feature_incompat; /* incompatible feature set */
__le32  s_feature_ro_compat;/* readonly-compatible feature set */
```

Drei Feature-Kategorien, an denen der Kernel entscheidet, ob und wie er dieses Dateisystem mounten wird:

- kompatibel: Dateisystem r/w mounten, auch wenn nicht unterst.
- ro-komp.: Dateisystem nur r/o mounten, wenn nicht unterst.
- inkompatibel: Dateisystem gar nicht mounten, wenn nicht unterst.

`s_feature_compat`, `s_feature_ro_compat`, `s_feature_incompat` sind Bitmaps, die die *in diesem Dateisystem* aktiven Features beschreiben, z. B.

```
#define EXT2_FEATURE_COMPAT_DIR_PREALLOC   0x0001
#define EXT2_FEATURE_COMPAT_IMAGIC_INODES  0x0002
#define EXT3_FEATURE_COMPAT_HAS_JOURNAL    0x0004
...
#define EXT2_FEATURE_INCOMPAT_COMPRESSION  0x0001
#define EXT2_FEATURE_INCOMPAT_FILETYPE     0x0002
#define EXT3_FEATURE_INCOMPAT_RECOVER      0x0004
...
```


Ext2-/Ext3-Superblock (5)

```
__u8    s_uuid[16];             /* 128-bit uuid for volume */
char    s_volume_name[16];      /* volume name */
char    s_last_mounted[64];     /* directory where last mounted */
__le32  s_algorithm_usage_bitmap; /* For compression */
/*
 * Performance hints.  Directory preallocation should only
 * happen if the EXT2_COMPAT_PREALLOC flag is on.
 */
__u8    s_prealloc_blocks;      /* Nr of blocks to try to preallocate */
__u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
```

Pre-allocation: Beim Reservieren eines Blocks ordnet Ext2 einer Datei nicht nur den angeforderten Block, sondern gleich mehrere „auf Vorrat“ zu: Das reduziert potenziell Fragmentierung und beschleunigt weitere Blockanforderungen, die kurz nach dieser folgen (etwa beim sequentiellen Schreiben einer Datei)

```
__u16   s_padding1;
```

Ext2-/Ext3-Superblock (6)

```
        /*
         * Journaling support valid if EXT3_FEATURE_COMPAT_HAS_JOURNAL set.
         */
        __u8    s_journal_uuid[16];  /* uuid of journal superblock */
        __u32   s_journal_inum;      /* inode number of journal file */
        __u32   s_journal_dev;       /* device number of journal file */
```

Das Journal kann auf einem anderen Dateisystem liegen; darum nicht nur Inode der Datei, sondern auch Gerätenummer

```
        __u32   s_last_orphan;       /* start of list of inodes to delete */
        __u32   s_hash_seed[4];      /* HTREE hash seed */
        __u8    s_def_hash_version;  /* Default hash version to use */
        __u8    s_reserved_char_pad;
        __u16   s_reserved_word_pad;
        __le32  s_default_mount_opts;
        __le32  s_first_meta_bg;     /* First metablock block group */
        __u32   s_reserved[190];     /* Padding to the end of the block */
};
```


Ext2-Inode (1)

Datei: *include/linux/ext2_fs.h*

```
/* Structure of an inode on the disk */
struct ext2_inode {
        __le16  i_mode;                 /* File mode */
        __le16  i_uid;                  /* Low 16 bits of Owner Uid */
        __le32  i_size;                 /* Size in bytes */
        __le32  i_atime;                /* Access time */
        __le32  i_ctime;                /* Creation time */
        __le32  i_mtime;                /* Modification time */
        __le32  i_dtime;                /* Deletion Time */
        __le16  i_gid;                  /* Low 16 bits of Group Id */
        __le16  i_links_count;          /* Links count */
        __le32  i_blocks;               /* Blocks count */
        __le32  i_flags;                /* File flags */
```

Flags:

```
        EXT2_SECRM_FL           0x00000001   secure deletion
        EXT2_UNRM_FL            0x00000002   record for undelete
        EXT2_COMPR_FL           0x00000004   compressed file
        EXT2_SYNC_FL            0x00000008   synchronous updates
        EXT2_IMMUTABLE_FL       0x00000010   immutable file
        EXT2_APPEND_FL          0x00000020   append only
        EXT2_NODUMP_FL          0x00000040   do not dump/delete file
        EXT2_NOATIME_FL         0x00000080   do not update .i_atime
        EXT2_DIRTY_FL           0x00000100   dirty (file is in use?)
        EXT2_COMPRBLK_FL        0x00000200   compressed blocks
        EXT2_NOCOMPR_FL         0x00000400   access raw compressed data
        EXT2_ECOMPR_FL          0x00000800   compression error
        EXT2_BTREE_FL           0x00010000   b-tree format directory
        EXT2_INDEX_FL           0x00010000   Hash indexed directory
        EXT3_JOURNAL_DATA_FL    0x00040000   journal file data
        EXT2_RESERVED_FL        0x80000000   reserved for ext2 implementation
```


Ext2-Inode (2)

```
        union {
                struct {
                        __le32  l_i_reserved1;
                } linux1;
            [...]
        } osd1;                                 /* OS dependent 1 */
        __le32  i_block[EXT2_N_BLOCKS];         /* Pointers to blocks */
        __le32  i_generation;                   /* File version (for NFS) */
        __le32  i_file_acl;                     /* File ACL */
        __le32  i_dir_acl;                      /* Directory ACL */

             #define i_size_high    i_dir_acl
```

Sollen Dateien eine 64-Bit-Länge haben, wird i_dir_acl (ansonsten unbenutzt für Dateien) als obere 32 Bit zusammen mit i_size genutzt

```
        __le32  i_faddr;                        /* Fragment address */
        union {
                struct {
                        __u8    l_i_frag;       /* Fragment number */
                        __u8    l_i_fsize;      /* Fragment size */
                        __u16   i_pad1;
                        __le16  l_i_uid_high;   /* these 2 fields    */
                        __le16  l_i_gid_high;   /* were reserved2[0] */
                        __u32   l_i_reserved2;
                } linux2;
            [...]
        } osd2;                                 /* OS dependent 2 */
};
```


Extra-Flags

- Ext2-/Ext3-Extra-Flags (immutable, append-only etc.) mit **chattr** bearbeiten

```
NAME
       chattr - change file attributes on a Linux second extended file system

SYNOPSIS
       chattr [ -RV ] [ -v version ] [ mode ] files...

DESCRIPTION
       chattr changes the file attributes on a Linux second extended file
       system.

       The format of a symbolic mode is +-=[ASacDdIijsTtu].

       The operator `+' causes the selected attributes to be added to the
       existing attributes of the files; `-' causes them to be removed; and
       `=' causes them to be the only attributes that the files have.
```

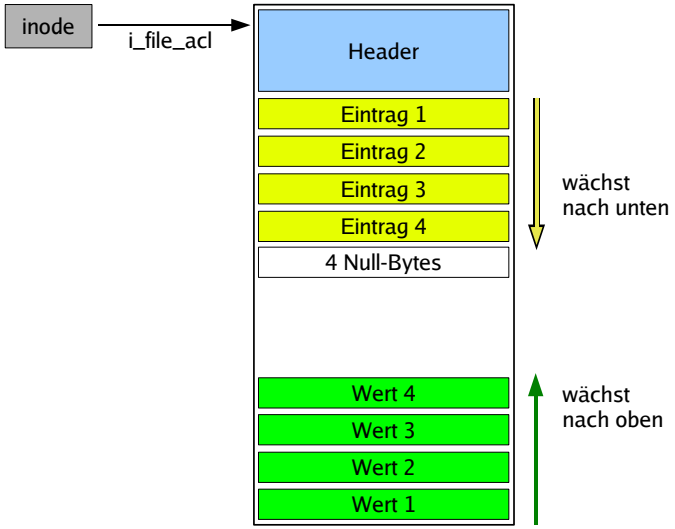
Erweiterte Attribute (1)

- Inode-Größe: 128 Byte
  - kein Platz für erweiterte Attribute
  - Vergrößerung auf 256 Byte nicht effizient
- Lösung: Separater Block für extended attributes (`ext2_xattr_entry`)
  - xattr-Header:

Datei: *fs/ext2/xattr.h*

```
struct ext2_xattr_header {
    __le32  h_magic;        /* magic number for identification */
    __le32  h_refcount;     /* reference count */
    __le32  h_blocks;       /* number of disk blocks used */
    __le32  h_hash;         /* hash value of all attributes */
    __u32   h_reserved[4];  /* zero right now */
};
```


Erweiterte Attribute (2)



Erweiterte Attribute (3)

xattr-Eintrag:

Datei: *fs/ext2/xattr.h*

```
struct ext2_xattr_entry {
    __u8    e_name_len;     /* length of name */
    __u8    e_name_index;   /* attribute name index */
    __le16  e_value_offs;   /* offset in disk block of value */
    __le32  e_value_block;  /* disk block attribute is stored on (n/i) */
    __le32  e_value_size;   /* size of attribute value */
    __le32  e_hash;         /* hash value of name and value */
    char    e_name[0];      /* attribute name */
};
```

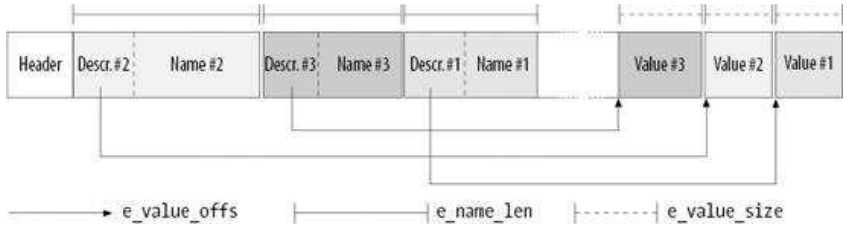


Bild: Understanding the Linux Kernel (Bovet/Cesati)

Erweiterte Attribute (4)

- bearbeiten mit **setfattr**, **getfattr**, **attr**:

```
amd64:/home/esser # setfattr -n user.foo -v betriebssysteme test.txt
amd64:/home/esser # getfattr -d test.txt
# file: test.txt
user.foo="betriebssysteme"

amd64:/home/esser # attr -g user.foo test.txt
Attribute "user.foo" had a 15 byte value for test.txt:
betriebssysteme
```

Dreierlei Attribute

Nicht verwechseln:

- Standard-Unix-Dateiattribute
  - UID, GID
  - Standardzugriffsrechte rwx,
  - Zugriffszeiten, ...
- Extra-Flags
  - immutable, compressed, secure deletion, ...
- Extended Attributes
  - beliebige, frei definierbare Attribute (inkl. ACLs)