9. Dateisysteme (5)

- **9.7 Standard-Dateisysteme: Ext2/Ext3 und NTFS**

/home/esser/Daten/Dozent/Folien/bs2-esser-11.odp

9.7.1 Standarddateisysteme: Linux Ext2 / Ext3

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

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)

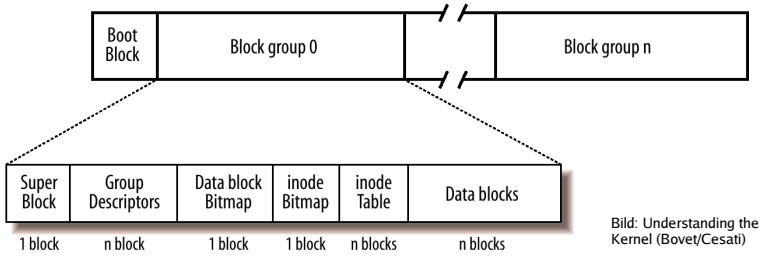
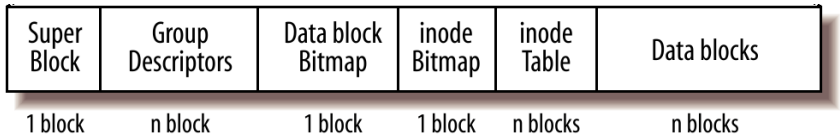


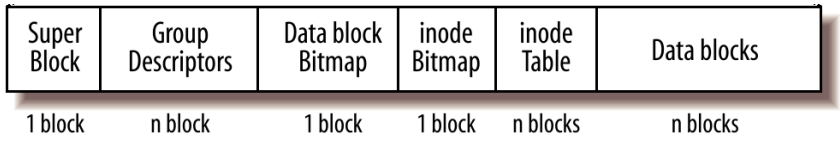
Bild: Understanding the Linux Kernel (Bovet/Cesati)

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^i, 5^i, 7^i (i>1)

Dateigrößen (1)

Flexibel auf zwei Arten:

- Beim Erstellen des Dateisystems unterschiedliche Blockgrößen möglich:
  - 1024-Byte-Blöcke -> max. Größe 16 GByte
  - 2048-Byte-Blöcke -> max. Größe 256 GByte
  - 4096-Byte-Blöcke -> max. Größe 4096 GByte (4 TB)

  (x 2 / x 16)
- Bei großen Dateien: Speichern der Blocknummern in Indirektionsblöcken (bis zu 3 Stufen):
  - Verdoppeln der Blockgröße = Ver-16-fachen (2^4) der Blocknummern: Faktor 2^3 aus Indirektion, Faktor 2 aus Blockgröße

Dateigrößen (2): Mehrfache Indirektion

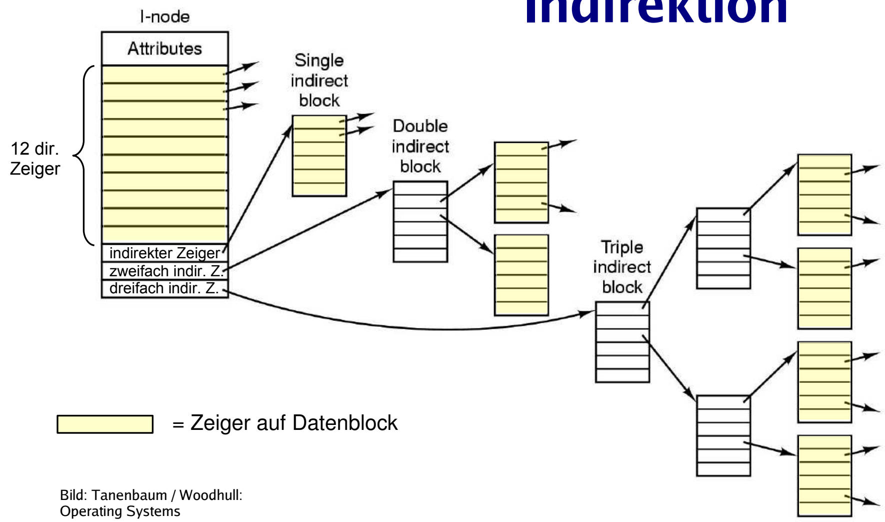


Bild: Tanenbaum / Woodhull: Operating Systems

Dateigrößen (3)

Rechenbeispiel

Länge einer (Platten-) Adresse: 4 Byte

Blockgröße BS:	**1024 Byte**	**2048 Byte**	**4096 Byte**
1 Block kann Adressen enthalten: BS/4=N	256	512	1024
12 dir. Zeiger: 12*BS =	12 KByte	24 KByte	48 KByte
1x indir. Z.: 1 Indir.-Bl.	256^1 x 1024 = 256 KB	512^1 x 2048 = 1 MB	1024^1 x 4096 = 4 MB
2x indir. Z.: N Indir.-Bl.	256^2 x 1024 = 64 MB	512^2 x 2048 = 512 MB	1024^2 x 4096 = 4 GB
3x indir. Z.: N^2 Indir.-Bl.	256^3 x 1024 = **16 GB**	512^3 x 2048 = **256 GB**	1024^3 x 4096 = **4 TB**

Angaben teilweise nur theoretisch; die maximale Dateigröße wird noch durch andere Faktoren begrenzt

Ext2-/Ext3-Superblock (1)

- Unterscheiden zwischen
  - Struktur auf der **Festplatte** (`ext2_super_block`)
  - Struktur im **Hauptspeicher** (`ext2_sb_info`)
- Auf Festplatte mit Typen fester Länge arbeiten, z. B.
  - `__u32`: unsigned 32 bit (ohne Vorzeichen)
  - `__s16`: signed 16 bit (mit Vorzeichen)
- Im Speicher mit nativen Datentypen arbeiten (`unsigned long`, `int` etc.)

Ext2-/Ext3-Superblock (2)

- Datenstrukturen auf Platte und im RAM

Type	Disk data structure	Memory data structure	Caching mode
Superblock	ext2_super_block	ext2_sb_info	Always cached
Group descriptor	ext2_group_desc	ext2_group_desc	Always cached
Block bitmap	Bit array in block	Bit array in buffer	Dynamic
inode bitmap	Bit array in block	Bit array in buffer	Dynamic
inode	ext2_inode	ext2_inode_info	Dynamic
Data block	Array of bytes	VFS buffer	Dynamic
Free inode	ext2_inode	None	Never
Free block	Array of bytes	None	Never

Tabelle: Understanding the Linux Kernel (Bovet/Cesati)

Ext2-/Ext3-Superblock (3)

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 (4)

```
    __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 (5)

```
__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 (6)

```
__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 (7)

```
__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 (8)

```
    /*
     * 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-Superblock im Speicher (1)

Datei: *include/linux/ext2_fs_sb.h*

```
/* second extended-fs super-block data in memory */
struct ext2_sb_info {
     unsigned long s_frag_size;          /* Size of a fragment in bytes */
     unsigned long s_frags_per_block;    /* Number of fragments per block */
     unsigned long s_inodes_per_block;   /* Number of inodes per block */
     unsigned long s_frags_per_group;    /* Number of fragments in a group */
     unsigned long s_blocks_per_group;   /* Number of blocks in a group */
     unsigned long s_inodes_per_group;   /* Number of inodes in a group */
     unsigned long s_itb_per_group;      /* Number of inode table blocks per group */
     unsigned long s_gdb_count;          /* Number of group descriptor blocks */
     unsigned long s_desc_per_block;     /* Number of group descriptors per block */
     unsigned long s_groups_count;       /* Number of groups in the fs */
     struct buffer_head * s_sbh;         /* Buffer containing the super block */
     struct ext2_super_block * s_es;     /* Pointer to the super block in the buffer */
     struct buffer_head ** s_group_desc;
     unsigned long  s_mount_opt;

           #define EXT2_MOUNT_CHECK         0x0001  /* Do mount-time checks */
           #define EXT2_MOUNT_OLDALLOC      0x0002  /* Don't use the new Orlov allocator */
           #define EXT2_MOUNT_GRPID         0x0004  /* Create files with directory's group */
           #define EXT2_MOUNT_DEBUG         0x0008  /* Some debugging messages */
           #define EXT2_MOUNT_ERRORS_CONT   0x0010  /* Continue on errors */
           #define EXT2_MOUNT_ERRORS_RO     0x0020  /* Remount fs ro on errors */
           #define EXT2_MOUNT_ERRORS_PANIC  0x0040  /* Panic on errors */
           #define EXT2_MOUNT_MINIX_DF      0x0080  /* Mimics the Minix statfs */
           #define EXT2_MOUNT_NOBH          0x0100  /* No buffer_heads */
           #define EXT2_MOUNT_NO_UID32      0x0200  /* Disable 32-bit UIDs */
           #define EXT2_MOUNT_XATTR_USER    0x4000  /* Extended user attributes */
           #define EXT2_MOUNT_POSIX_ACL     0x8000  /* POSIX Access Control Lists */
```


Ext2-Superblock im Speicher (2)

```
     uid_t s_resuid;
     gid_t s_resgid;
     unsigned short s_mount_state;
     unsigned short s_pad;
     int s_addr_per_block_bits;
     int s_desc_per_block_bits;
     int s_inode_size;
     int s_first_ino;
     spinlock_t s_next_gen_lock;
     u32 s_next_generation;
     unsigned long s_dir_count;
     u8 *s_debts;
     struct percpu_counter s_freeblocks_counter;
     struct percpu_counter s_freeinodes_counter;
     struct percpu_counter s_dirs_counter;
     struct blockgroup_lock s_blockgroup_lock;
};
```


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 (vgl. *9. Dateisysteme(2)/Folie 21*)

```
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.
```

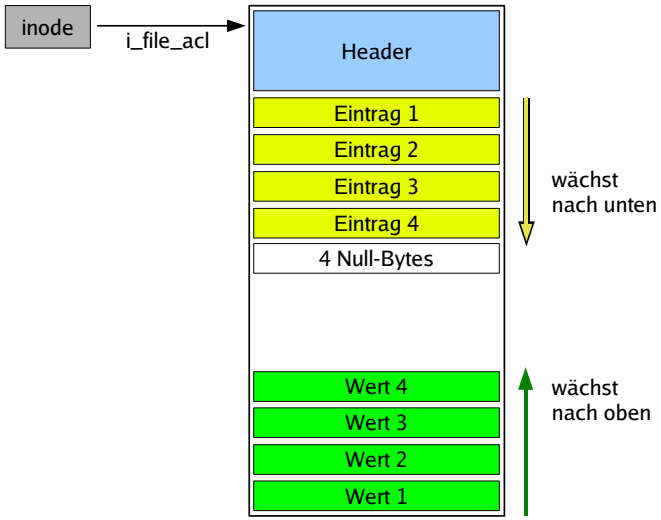

Erweiterte Attribute (1)

- 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 */
};
```

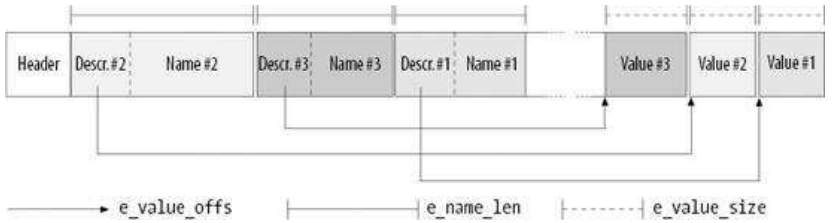


Bild: Understanding the Linux Kernel (Bovet/Cesati)

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)

Journaling (1)

- Transaktionskonzept ursprünglich von Datenbanken
- Journaling-Mechanismus für Dateisysteme vereinfacht (Performance und Implementations-komplexität).
- JFS zerlegt Dateisystemmodifikationen in **atomare Operationen** und gruppiert diese zu **Transaktionen**.
- Änderungen werden in **Journal** geloggt

Die Folien zum Journaling basieren auf folgendem Foliensatz:
Mengjun Xie, „The Ext3 File System", http://www.cs.wm.edu/~kearns/780S04/slides/ext3.pdf

Journaling (2)

- Alle Änderungen auf einem JFS in zwei Stufen durchführen:

1. Commit
   - Einzelne / mehrere Disk-Updates bilden eine Transaktion
   - Transaktion *ins Journal* (nicht ins FS) schreiben
   - Erst wenn ein „commit block" geschrieben wurde, ist die Transaktion abgeschlossen
   - Journal-Blöcke erst recyceln, wenn alle Daten im Dateisystem stehen (sync)
2. Check-point
   - Transaktion aus dem Journal ins Dateisystem übertragen (flush)
   - Journal-Speicherplatz freigeben

Journaling (3)

- Da Journaling teuer ist, schreiben die meisten JFS nur (Änderungen von) Metadaten ins Journal.
- Journaling sorgt für Konsistenz:
  - Commit-Phase ist atomar. Neue Daten werden ins Journal „committed“, alte stehen noch im Dateisystem.
  - Nach einem Absturz kann fsck zwischen zwei Fällen unterscheiden:
    - Fehler trat vor Commit auf: Änderung geht zwar verloren (wird verworfen), aber Dateisystem ist konsistent, weil noch keine Flush-Operation gestartet wurde.
    - Fehler trat nach Commit auf: Änderung ist gültig und kann angewendet werden (flush).

Journaling (4)

- (Konsistenz ...)
  - Ext3 garantiert Konsistenz nur auf System-Call-Ebene. Darum gibt es für Operationen, die mehrere System Calls benötigen, keine Konsistenzgarantie.
  - Die meisten JFS legen keine Journal-Einträge für Operationen auf Dateidaten-Blöcken an
    ➞ Damit gibt es keinen Schutz vor Korruption einzelner Dateien (nur die Metadaten sind geschützt).

Journaling (5)

- Ext3-Journal:
  - Journal liegt meist in einer versteckten Datei *.journal* im Wurzelverzeichnis des Datei-systems. Ext3 kümmert sich nicht selbst um das Journaling – das macht ein allgemeiner Kernel-Layer, das **Journaling Block Device** (JBD)

```
amd64:/home/esser # modinfo ext3
filename:       /lib/modules/2.6.11.4-20a-default/kernel/fs/ext3/ext3.ko
description:    Second Extended Filesystem with journaling extensions
license:        GPL
vermagic:       2.6.11.4-20a-default 586 REGPARM gcc-3.3
depends:        jbd

amd64:/home/esser # modinfo jbd
filename:       /lib/modules/2.6.11.4-20a-default/kernel/fs/jbd/jbd.ko
license:        GPL
vermagic:       2.6.11.4-20a-default 586 REGPARM gcc-3.3
depends:
```


Ext3 und JBD (1)

- Zwei separate Ebenen
  - **fs/ext3:** Dateisystem mit Transaktionen
  - **fs/jbd:** Journaling-Code.
- Ext3 ruft JBD auf, wenn es nötig ist:
  - Anfang/Ende einer Transaktion
  - Journal-Wiederherstellung nach unsauberem Reboot
- Verbund-Transaktionen
  - Eine Transaktion kann mehrere Updates enthalten

Ext3 und JBD (2)

- JBD ist vollständig unabhängig von Ext3. Es ist eine abstrakte Journaling-Schicht, mit der Änderungen an beliebigen Blockgeräten einer Transaktionssemantik gehorchen.
- Die Zusammenarbeit von Ext3 und JBD basiert im Wesentlichen auf drei fundamentalen Einheiten:
  - log record
  - atomic operation handle
  - Transaktion

Ext3 und JBD (3)

- **Log record:** beschreibt ein einzelnes Update eines Plattenblocks des JFS.
  - Log records werden im Journal als normale Daten- (oder Metadaten-) Blöcke repräsentiert.
  - Zu jedem solchen Block gehört ein kleines Tag vom Typ *journal_block_tag_t*, das die logische Blocknummer des Blocks im Dateisystem und ein paar Status-Flags enthält.

Ext3 und JBD (4)

- **Atomic operation handle:** enthält log records die zu einer einzelnen High-Level-Änderung am Dateisystem gehören
  - typischerweise erzeugt jeder System Call, der das Dateisystem modifiziert, einen einzelnen atomic operation handle.
  - Jeder atomic operation handle wird durch einen Deskriptor vom Typ *handle_t* dargestellt.

Ext3 und JBD (5)

- **Transaktion:** umfasst mehrere atomic operation handles, deren log records gleichzeitig als „valid“ markiert werden.
  - Alle log records einer Transaktion stehen in aufeinanderfolgenden (zusammenhängenden) Blöcken im Journal.
    Eine Transaktion wird durch einen Deskriptor vom Typ *transaction_t* dargestellt. Dessen wichtigstes Feld ist *t_state*, das den aktuellen Status der Transaktion enthält.

Ext3 Journaling Modes

Ext3 lässt sich so einstellen, dass es Operationen loggt, die Filesystem-Metadaten, aber auch Datei-Datenblocks betreffen. Dazu gibt es drei Journaling-Betriebsarten:

- **Journal:** Alle Änderungen von Dateisystemdaten- und -metadaten im Journal loggen.
  (Sicherster und langsamster Ext3 Journaling Mode)
- **Ordered:** Nur Änderungen an Metadaten loggen. Ext3 ordnet Metadaten und Datenblöcke aber so an, dass Datenblöcke vor den Metadaten auf Platte geschrieben werden.
  (Standard Ext3 Journaling Mode)
- **Writeback:** Nur Änderungen an Metadaten loggen – ohne die Anordnung der Ordered-Variante.
  (Schnellster Ext3 Journaling Mode)

Vorschau

Vorlesung 16.01.2007

letztes Teilkapitel 9.7.2:

Windows
New Technology Filesystem (NTFS)