9. Dateisysteme (2)

- 9.3 Virtuelle FS
  - 9.3.1 Linux VFS
  - 9.3.2 Windows IFS
- 9.4 Dateizugriff in Linux-Programmen

9.3 Virtuelle Dateisysteme

Virtuelles Dateisystem – VFS (1)

Zwei Schichten einführen

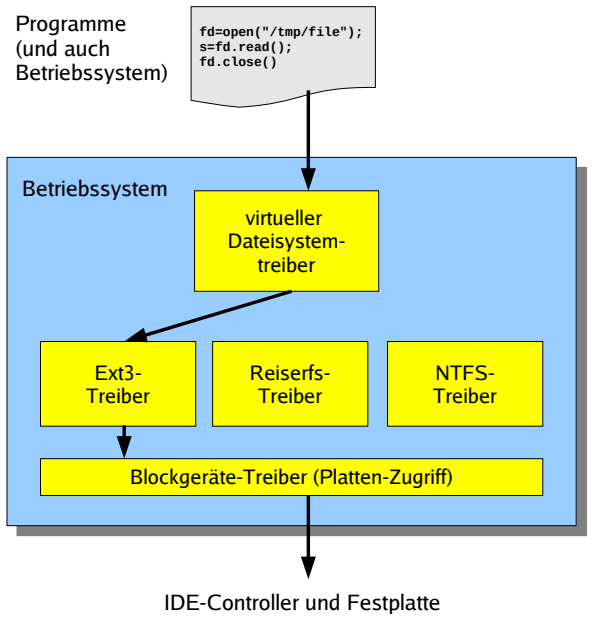
- VFS-Treiber stellt High-Level-Dateioperationen bereit (create, delete, rename, open, close, read, write, seek, link, ...)
- Kommunikation aller Programme (und auch des BS selbst) nur mit dem VFS-Treiber
- VFS-Treiber leitet Anfragen an Spezialtreiber für die Dateisysteme weiter
- Spezialtreiber beherrschen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

VFS (5): Standard-Funktionen

- Das VFS kennt die Datei-Attribute und -Operationen, die für das Betriebssystem relevant sind
- Unterscheidung (hier) zwischen
  - **Linux VFS**
  - OS/2 und **Windows IFS** (**I**nstallable **F**ile **S**ystem)

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)**

Linux VFS
Standard-Funktionen (2)

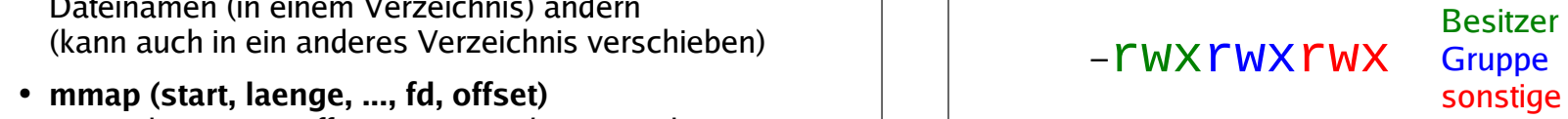
- **stat (filename, status)**
  fstat (fd, status)
  Informationen über Datei abrufen
- **unlink (name)** Eintrag in einem Verzeichnis löschen – evtl. auch die Datei, wenn dies der letzte Link war
- **rename (oldpath, newpath)**
  Dateinamen (in einem Verzeichnis) ändern
  (kann auch in ein anderes Verzeichnis verschieben)
- **mmap (start, laenge, ..., fd, offset)**
  Datei ab Position *offset* mit Länge *laenge* in den Hauptspeicher einblenden

Linux VFS
Standard-Funktionen (3)

- **mkdir (pathname, mode)**
  Verzeichnis erzeugen
- **rmdir (pathname)**
  leeres Verzeichnis löschen
- **chdir (pathname)**
  Arbeitsverzeichnis wechseln
- **getcwd (*buf, size)**
  Name des Arbeitsverzeichnis in Puffer schreiben

Linux-Dateiattribute (1)

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



- Anwender können zu verschiedenen Gruppen gehören

Linux-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

Linux-Dateiattribute (3)

- Beim Erzeugen einer Datei werden Standard-rechte 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 2006-12-04 20:48 Datei
$ umask u=rw,g=r,o=
$ umask
0137
$ touch Test; ls -l Test
-rw-r-----  1 esser users 0 2006-12-04 20:50 Test
```


Linux-Dateiattribute (4)

- umask wird von 777 (Maximalrechte) bitweise abgezogen, um konkrete Dateirechte zu berechnen;
- Ausführrecht wird beim Erzeugen einer Datei nie vergeben

Linux-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

Linux-Dateiattribute (6)

- Erweiterte Attribute (nur ext2, ext3)
  - append only (**a**): auf Datei darf nur im Append-Modus geschrieben werden
  - auto-compression (**c**): Datei wird automatisch komprimiert
  - immutable (**i**): Datei darf nicht verändert werden
  - secure deletion (**s**): Wird diese Datei gelöscht, wird der Inhalt vorher mit 0-Bytes überschrieben
  - undeletion (**u**): Wird die Datei gelöscht, kann das rückgängig gemacht werden

Linux-Dateiattribute (7)

ACL (Access Control Lists)

- feineres Zugriffs-Tuning mit **setfacl**
- für zusätzliche Benutzer und Gruppen Rechte separat festlegen, z. B.:

```
setfacl -m u:benutzer:r datei
```


Windows IFS (1)

- Ähnliches Konzept wie unter Linux
- IFS-Treiber für verschiedene konkrete Dateisysteme
- Eine Schicht höher: generische Betriebssystemfunktionen für Zugriffe auf Dateisystem

Windows IFS (2)

- Standard-IFS-Treiber
  - NTFS (New Technology File System)
  - FAT (DOS File Allocation Table; FAT12, -16, -32)
  - CDFS (mit „Joliet"-Erweiterung) und UDF
  - ehemals auch HPFS (OS/2 High Performance FS)
- IFS-Treiber von anderen Anbietern
  - ext2ifs (Linux ext2 / ext3)
  - ReiserDriver (Linux ReiserFS)
  - HFS IFS (Apple HFS)
  - „Paragon Alles Mounter" (ext2/3)

Windows IFS (3)

- Filesystem Filter Driver
  - optionale Zusatztreiber, die es erlauben, das Verhalten eines Dateisystems zu verändern
  - mögliche Anwendungen:
    - On-the-fly-Virenfilter
    - Transparente Verschlüsselung
    - Backup-Mechanismen
    - Überwachung
  - Beispiel: Sysinternals FileMon für Überwachung der Dateisystemzugriffe

Windows-Standardfunktionen (1)

- **handle = CreateFile (filename, access, ...)**
  Datei erzeugen oder vorhandene Datei öffnen
- **DeleteFile (filename)**
  Datei löschen
- **CloseHandle (handle)**
  Datei schließen
- **ReadFile (handle, buffer, len, &count, NULL)**
  aus geöffneter Datei lesen
- **WriteFile (handle, buffer, len, &count, NULL)**
  in geöffnete Datei schreiben

Windows-Standardfunktionen (2)

- **SetFilePointer (handle, offset, 0, method)**
  an bestimmte Stelle in der Datei springen
  (abhängig vom letzten Parameter: ab Dateianfang, Dateiende oder ab aktueller Position)
- **GetFileAttributes (filename)**
  Dateiattribute abfragen
- **LockFile (handle, offset, length)**
  Blöcke in der Datei gegen parallelen Zugriff sperren
- **UnlockFile (handle, offset, length)**
  Sperre aufheben

Windows-Standardfunktionen (3)

- **CreateDirectory ()** neues Verzeichnis erzeugen
- **RemoveDirectory ()** leeres Verzeichnis löschen
- **FindFirstFile ()** ersten Eintrag in Verzeichnis lesen
- **FindNextFile ()** nächsten Eintrag lesen
- **MoveFile ()** Datei in anderes Verzeichnis verschieben
- **SetCurrentDirectory ()** Arbeitsverzeichnis wechseln

Windows-Dateiattribute (1)

- **FILE_ATTRIBUTE_ARCHIVE**
  Datei soll archiviert werden
- **FILE_ATTRIBUTE_ENCRYPTED:** Datei ist verschlüsselt
- **FILE_ATTRIBUTE_HIDDEN**
  Datei nicht in Standard-Listings anzeigen
- **FILE_ATTRIBUTE_OFFLINE**
  Datei liegt in Offline-Speicher (z.B. Magnetband)
- **FILE_ATTRIBUTE_READONLY:** nur-lesbar
- **FILE_ATTRIBUTE_SYSTEM:** Systemdatei
- **FILE_ATTRIBUTE_TEMPORARY:** temporäre Datei
- **FILE_ATTRIBUTE_NORMAL:** keine Attribute

Windows-Dateiattribute (2)

Erweiterung der klassischen
FAT-Dateiattribute:

- archive (A)
- read-only (R)
- hidden (H)
- system (S)

9.4 Praxis: Dateizugriff in Linux-Programmen

Dateizugriff in Linux-Programmen (1)

Datei öffnen, lesen, schreiben, schließen

(hatten wir schon in *Betriebssysteme I*)

```
int fd = open ( "/etc/fstab", O_RDONLY );
  while ( (len = read ( fd, line, bufsiz )) > 0 ) {
    if ( len < bufsiz) { line[len]='\0'; }
    printf ("%s", line );
  }
  close (fd);
```

Jetzt mehr Details ...

Dateizugriff in Linux-Programmen (2)

fd = open (filename,flags);

Mögliche Flags:

- O_RDONLY: Nur zum Lesen öffnen
- O_WRONLY: Nur zum Schreiben öffnen
- O_RDWR: Zum Schreiben und Lesen öffnen
- O_CREAT: Datei erzeugen, wenn sie noch nicht exisitiert
- O_TRUNC: Wenn Datei schon existiert, überschreiben (Länge auf 0 setzen)
- O_APPEND: Append-Modus; Dateizeiger auf Dateiende posititionieren

Dateizugriff in Linux-Programmen (3)

- O_SYNC: Synchroner I/O-Modus (jeden Schreibbefehl sofort ausführen)
- O_NOATIME: Bei Lesezugriffen nicht die Access-Time aktualisieren
- O_LARGEFILE: Datei benötigt 64 Bit zur Größenangabe
- O_NOFOLLOW: Wenn der Dateiname ein symbolischer Link ist, fehlschlagen

- und diverse weitere Flags ...

Dateizugriff in Linux-Programmen (4)

int stat(const char *filename, struct stat *buf);

gibt zu einer Datei folgende Eigenschaften zurück:

```
struct stat
{
  dev_t          st_dev;      /* Device (welches Dateisystem?) */
  ino_t          st_ino;      /* INode */
  mode_t         st_mode;     /* Zugriffsrechte */
  nlink_t        st_nlink;    /* Anzahl harter Links */
  uid_t          st_uid;      /* UID des Besitzers */
  gid_t          st_gid;      /* GID des Besitzers */
  dev_t          st_rdev;     /* Gerätetyp (wenn INode-Gerät) */
  off_t          st_size;     /* Größe in Bytes */
  unsigned long  st_blksize;  /* Blockgröße */
  unsigned long  st_blocks;   /* Allozierte Blocks (512-Byte-Blocks) */
  time_t         st_atime;    /* Letzter Zugriff */
  time_t         st_mtime;    /* Letzte Modifikation */
  time_t         st_ctime;    /* Letzte Änderung (von Verwaltungsinf.) */
};
```


Dateizugriff in Linux-Programmen (5)

Beispiel für die Verwendung von **stat( )**:

```
/*  fileinfo.c  */
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

main () {
  struct stat status;
  int rdev;

  if (stat("/etc/fstab", &status) == -1) {
    return -1;
  } else {
    printf ("Dateigroesse:  %d \n", status.st_size);
    printf ("UID:           %d \n", status.st_uid);
    printf ("GID:           %d \n", status.st_gid);
    rdev = status.st_rdev;
    printf ("Geraetedatei:  (%d,%d) \n", rdev/256, rdev%256);
  }
}
```

Dateizugriff in Linux-Programmen (6)

```
$ ls -l /etc/fstab
-rw-r--r--  1 root root 992 2005-04-11 20:24 /etc/fstab
$ fileinfo /etc/fstab
Dateigroesse:  992
UID:           0
GID:           0
Geraetedatei:  (0,0)

$ ls -l /dev/sda3
brw-rw----  1 root disk 8, 3 2005-03-19 20:36 /dev/sda3
$ fileinfo /dev/sda3
Dateigroesse:  0
UID:           0
GID:           6
Geraetedatei:  (8,3)

$ ls -l /dev/tty22
crw--w----  1 root tty 4, 22 2005-03-19 20:36 /dev/tty22
$ fileinfo /dev/tty22
Dateigroesse:  0
UID:           0
GID:           5
Geraetedatei:  (4,22)
```


Dateizugriff in Linux-Programmen (7)

Zugriffsrechte prüfen (1)

```
S_IFMT     0017000   Bitmaske für die Dateityp-Bitfelder
S_IFSOCK   0140000   Socket
S_IFLNK    0120000   symbolische Verknüpfung
S_IFREG    0100000   reguläre Datei
S_IFBLK    0060000   blockorientiertes Gerät
S_IFDIR    0040000   Verzeichnis
S_IFCHR    0020000   zeichenorientiertes Gerät
S_IFIFO    0010000   FIFO
S_ISUID    0004000   SUID-Bit
S_ISGID    0002000   SGID-Bit (siehe unten)
S_ISVTX    0001000   Sticky-Bit (siehe unten)
S_IRWXU    00700     Bitmaske für Besitzerzugriffsrechte
S_IRUSR    00400     Besitzer hat Lesezugriff
S_IWUSR    00200     Besitzer hat Schreibzugriff
S_IXUSR    00100     Besitzer hat Ausführungsrechte
S_IRWXG    00070     Bitmaske für Gruppenzugriffsrechte
S_IRGRP    00040     Gruppe hat Lesezugriff
S_IWGRP    00020     Gruppe hat Schreibzugriff
S_IXGRP    00010     Gruppe hat Ausführungsrechte
S_IRWXO    00007     Bitmaske für Zugriffsrechte Anderer (nicht in Gruppe)
S_IROTH    00004     Andere haben Lesezugriff
S_IWOTH    00002     Andere haben Schreibzugriff
S_IXOTH    00001     Andere haben Ausführungsrechte
```


Dateizugriff in Linux-Programmen (8)

Zugriffsrechte prüfen (2)

```
struct stat status;
mode_t modus;
stat("/etc/fstab", &status);
modus = status.st_mode;
if ( modus & S_IFREG ) { printf ("%s", "Reguläre Datei \n"); }
if ( modus & S_IFDIR ) { printf ("%s", "Verzeichnis \n"); }
if ( modus & S_IFLNK ) { printf ("%s", "Symbolischer Link \n"); }
```

```
$ testfile /etc/fstab
Reguläre Datei
$ testfile /etc
Verzeichnis
$ testfile /etc/rc.d
Symbolischer Link
$ testfile /dev/zero
$
```