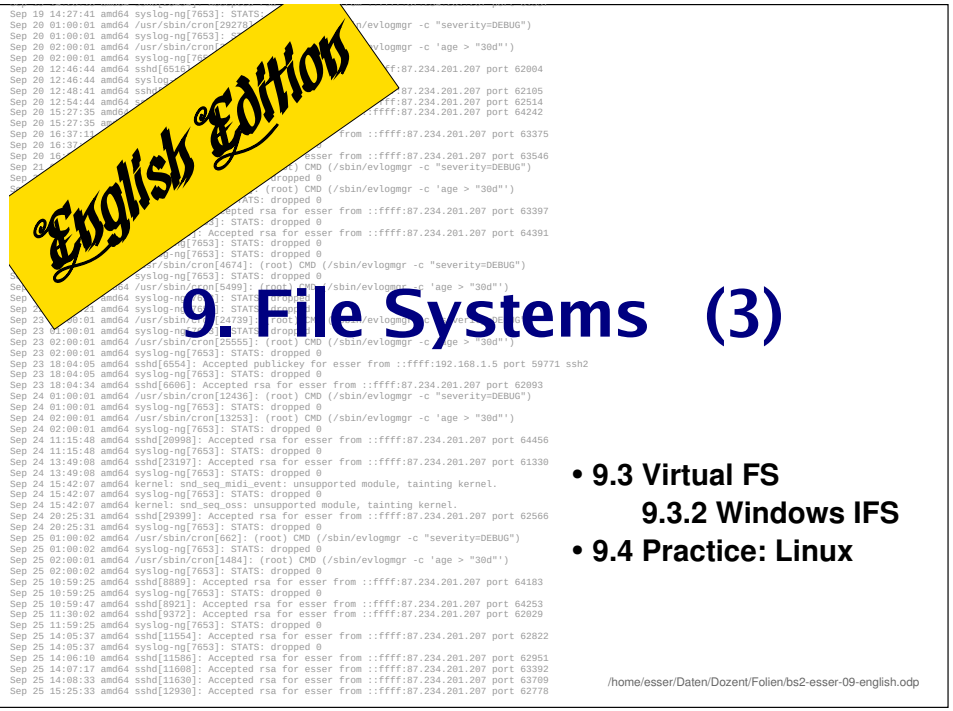
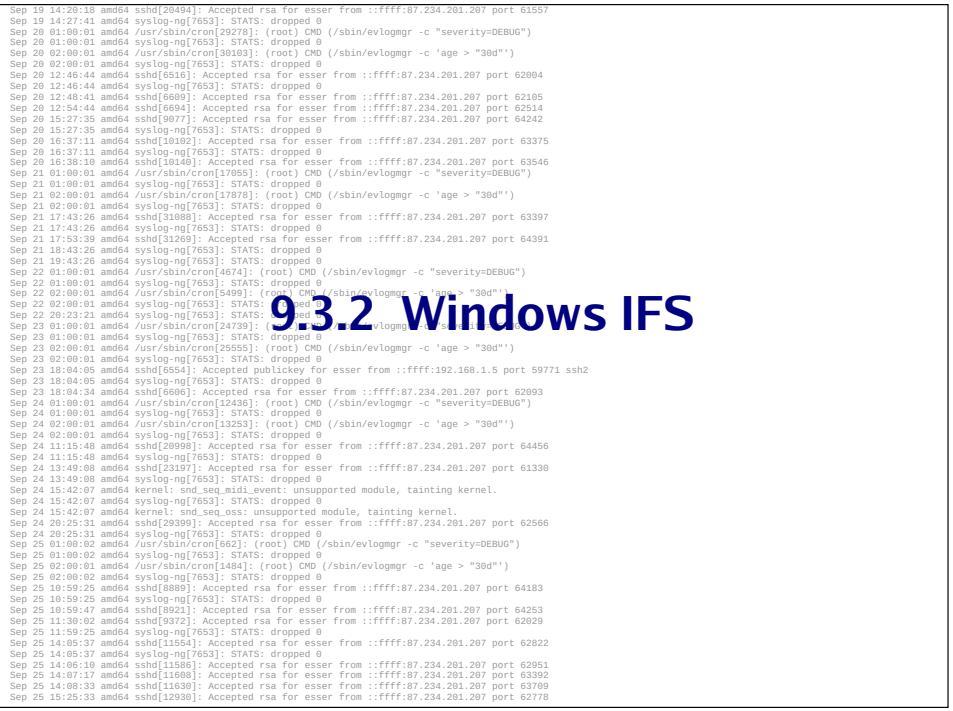


9. File Systems (3)

- 9.3 Virtual FS
  - 9.3.2 Windows IFS
- 9.4 Practice: Linux

Windows IFS (1)

- concept is similar to that of Linux
- IFS drivers for several specific file systems
- one level above: generic operating system functions for accessing a file system



9.3.2 Windows IFS

Windows IFS (2)

- Standard IFS drivers
  - NTFS (New Technology File System)
  - FAT (DOS File Allocation Table; FAT12, -16, -32)
  - CDFS (with „Joliet“ extension) and UDF
  - old versions: HPFS (OS/2 High Performance FS)
- IFS drivers from third parties
  - ext2ifs (Linux ext2 / ext3)
  - ReiserDriver (Linux ReiserFS)
  - HFS IFS (Apple HFS)
  - „Paragon Alles Mounter“ (ext2/3)

Windows IFS (3)

- Filesystem Filter Driver
  - optional add-on drivers which can modify a file system's behavior
  - possible applications:
    - on-the-fly virus filter
    - transparent encryption
    - backup mechanisms
    - monitoring
  - Example: Sysinternals' FileMon for monitoring file system access

Windows-Standardfunktionen (1)

- **handle = CreateFile (filename, access, ...)**
  create and open file or open existing file
- **DeleteFile (filename)**
  delete file
- **CloseHandle (handle)**
  close file
- **ReadFile (handle, buffer, len, &count, NULL)**
  read from open file
- **WriteFile (handle, buffer, len, &count, NULL)**
  write to open file

Windows-Standardfunktionen (2)

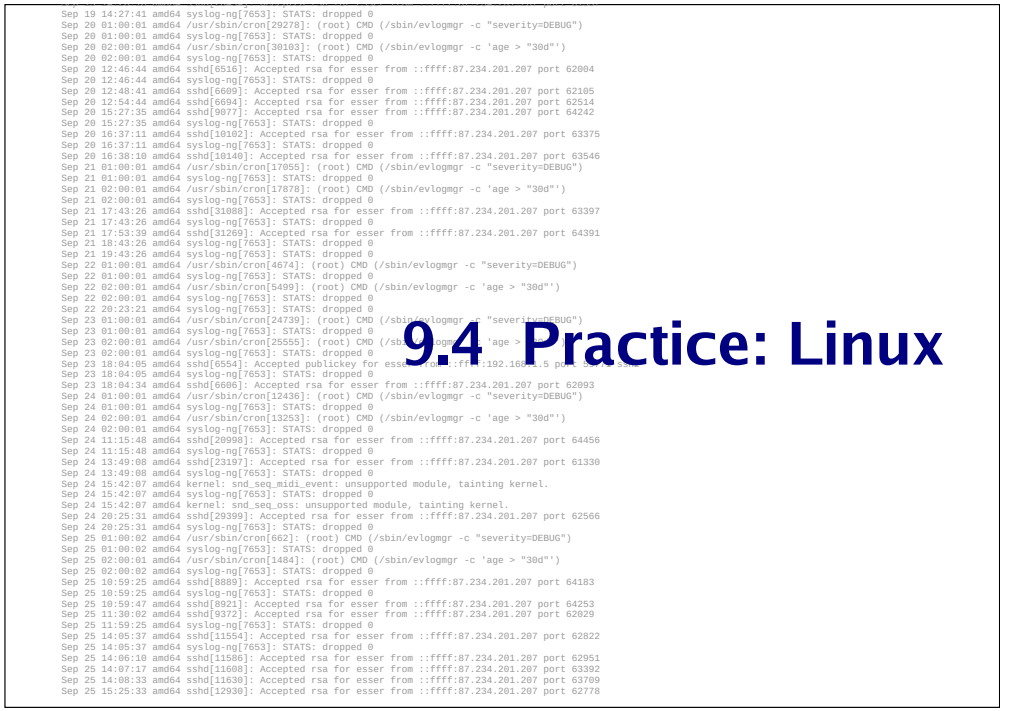
- **SetFilePointer (handle, offset, 0, method)**
  jump to a specific offset inside the file
  (depending on the last parameter: from 0, from end of file or relative from the current position)
- **GetFileAttributes (filename)**
  query file attributes
- **LockFile (handle, offset, length)**
  lock blocks in the file against concurrent access
- **UnlockFile (handle, offset, length)**
  release lock

Windows-Standardfunktionen (3)

- **CreateDirectory ()** create a new directory
- **RemoveDirectory ()** delete an empty directory
- **FindFirstFile ()** read first entry of a directory
- **FindNextFile ()** read next entry
- **MoveFile ()** move file into a different directory or rename it
- **SetCurrentDirectory ()** change working directory

Windows File Attributes (1)

- **FILE_ATTRIBUTE_ARCHIVE**
  file should be archived (with next backup operation)
- **FILE_ATTRIBUTE_ENCRYPTED:** file is encrypted
- **FILE_ATTRIBUTE_HIDDEN**
  don't list file in standard directory listings
- **FILE_ATTRIBUTE_OFFLINE**
  file is in offline storage (e.g. magnetic tape)
- **FILE_ATTRIBUTE_READONLY:** read-only
- **FILE_ATTRIBUTE_SYSTEM:** system file
- **FILE_ATTRIBUTE_TEMPORARY:** temporary file
- **FILE_ATTRIBUTE_NORMAL:** no attributes



9.4 Practice: Linux

Windows File Attributes (2)

Extension of classic **FAT file attributes:**

- archive (A) -> **FILE_ATTRIBUTE_ARCHIVE**
- read-only (R) -> **FILE_ATTRIBUTE_READONLY**
- hidden (H) -> **FILE_ATTRIBUTE_HIDDEN**
- system (S) -> **FILE_ATTRIBUTE_SYSTEM**

9.4.1 File Access in Linux Programs

File Access in Linux Programs (1)

open and close file, read and write

(we saw that in *Operating Systems I*)

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

fd = open ( "/etc/fstab", O_WRONLY );
write ( fd, buffer, bufsiz );
```

now: more details ...

File Access in Linux Programs (2)

fd = open (filename,flags);

possible flags:

- O_RDONLY: open for read access only
- O_WRONLY: open for write access only
- O_RDWR: open for reading and writing
- O_CREAT: create file, if it does not yet exist
- O_TRUNC: if file already exists, overwrite (truncate) it (set file length to 0)
- O_APPEND: append mode; file pointer is positioned at end of file

File Access in Linux Programs (3)

- O_SYNC: synchronous I/O mode (execute each write command immediately)
- O_NOATIME: don't update access time after read access
- O_LARGEFILE: file needs 64 bits for size field
- O_NOFOLLOW: fail if the filename is (contains) a symbolic link

- and several further flags ...

File Access in Linux Programs (4)

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

will return the following properties of a file:

```
struct stat
{
  dev_t          st_dev;      /* Device (which filesystem?) */
  ino_t          st_ino;      /* INode */
  mode_t         st_mode;     /* access rights */
  nlink_t        st_nlink;    /* number of (hard) links = dir. entries */
  uid_t          st_uid;      /* UID of the owner */
  gid_t          st_gid;      /* GID of the owning group */
  dev_t          st_rdev;     /* device type (when INode device) */
  off_t          st_size;     /* size in bytes */
  unsigned long  st_blksize;  /* block size */
  unsigned long  st_blocks;   /* allocated blocks (512 byte blocks) */
  time_t         st_atime;    /* last access */
  time_t         st_mtime;    /* last modification */
  time_t         st_ctime;    /* last change (of inode information) */
};
```

File Access in Linux Programs (5)

example for using **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 ("File size:     %d \n", status.st_size);
    printf ("UID:           %d \n", status.st_uid);
    printf ("GID:           %d \n", status.st_gid);
    rdev = status.st_rdev;
    printf ("device file:   (%d,%d) \n", rdev/256, rdev%256);
  }
}
```


File Access in Linux Programs (6)

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

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

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

excerpt from */etc/group*:

```
root:x:0:
bin:x:1:daemon
daemon:x:2:
sys:x:3:
tty:x:5:
disk:x:6:
lp:x:7:
www:x:8:
kmem:x:9:
wheel:x:10:
mail:x:12:
```

first line of */etc/passwd*:

```
root:x:0:0:
  root:/root:
  /bin/bash
```


File Access in Linux Programs (7)

check access rights and properties (1)

```
S_IFMT     0017000   Bitmask for filetype bit fields
S_IFSOCK   0140000   socket
S_IFLNK    0120000   symbolic link
S_IFREG    0100000   regular file
S_IFBLK    0060000   block-oriented device
S_IFDIR    0040000   directory
S_IFCHR    0020000   character-oriented files
S_IFIFO    0010000   FIFO
S_ISUID    0004000   SUID bit
S_ISGID    0002000   SGID bit
S_ISVTX    0001000   Sticky bit
S_IRWXU      00700   Bitmask for owner access rights
S_IRUSR      00400   owner has read access
S_IWUSR      00200   owner has write access
S_IXUSR      00100   owner has execute access
S_IRWXG      00070   Bitmask for group access rights
S_IRGRP      00040   group has read access
S_IWGRP      00020   group has write access
S_IXGRP      00010   group has execute access
S_IRWXO      00007   Bitmask for access rights of others (neither group now owner)
S_IROTH      00004   other have read access
S_IWOTH      00002   other have write access
S_IXOTH      00001   other have execute access
```


File Access in Linux Programs (8)

check access rights and properties (2)

```
struct stat status;
mode_t modus;
stat ("/etc/fstab", &status);
modus = status.st_mode;
if ( modus & S_IFREG ) { printf ("%s", "regular file \n"); }
if ( modus & S_IFDIR ) { printf ("%s", "directory \n"); }
if ( modus & S_IFLNK ) { printf ("%s", "symbolic link \n"); }


$ testfile /etc/fstab
regular file
$ testfile /etc
directory
$ testfile /etc/rc.d
symbolic link
$ testfile /dev/zero
$
```

File Access in Linux Programs (9)

read or write a specific block

block size: *bsize*, n^{th} Block (starting with 0)

```
int bsize;
char data[bsize];

fd = open (filename, O_RDONLY);
lseek (fd, bsize*n, SEEK_SET);
read (fd, &data, bsize);


fd = open (filename, O_WRONLY);
lseek (fd, bsize*n, SEEK_SET);
write (fd, &data, bsize);


lseek (fd, bsize*n, SEEK_CUR);  /* jump n blocks, relatively */
```


File Access in Linux Programs (10)

query current read/write position in file

```
pos = lseek (fd, 0, SEEK_CUR);
  /* jumps relatively 0 bytes, i.e. not at all      */
  /* lseek() always return new position after
     the jump                                       */
```


File Access in Linux Programs (11)

working with blocks: a mini database

```
struct dbrecord {
  int personid;      /* ID for the person               */
  int flag;          /* Flag: this entry is in use      */
  char nachname[40]; /* last name                       */
  char vorname[40];  /* first name                      */
  char strasse[50];  /* address                         */
  char plz[5];
  char ort[40];
};
struct dbrecord dr;
const int recsize = sizeof (struct dbrecord);  /* 184 */

struct dbrecord dummy_record (int i) {
  struct dbrecord d;
  d.personid=i; d.flag=0;
  strcpy (d.nachname,"Mustermann");
  strcpy (d.vorname,"Thomas");
  strcpy (d.strasse,"Lothstrasse");
  strcpy (d.plz,"80000");
  strcpy (d.ort,"Muenchen");
  return d;
}
```


File Access in Linux Programs (12)

```
void init_db () {
  int fd, i;
  fd = open ("/tmp/mytmpfile",
     O_WRONLY|O_CREAT,S_IREAD|S_IWRITE);
  for (i=0; i<20; i++) {
    dr = dummy_record (i);
    write (fd, &dr, recsize);
  }
  close (fd);
}

void read_record (int i) {
  struct dbrecord dr;
  int fd;
  fd = open ("/tmp/mytmpfile",O_RDONLY);
  lseek (fd, i*recsize, SEEK_SET);
  read (fd, &dr, recsize);
  printf ("ID: %d, Flag: %d \n",
          dr.personid, dr.flag);
  printf ("last name:  %s \n",dr.nachname);
  printf ("first name: %s \n",dr.vorname);
  printf ("address:    %s \n",dr.strasse);
  printf ("town:       %s \n\n",dr.ort);
  close (fd);
  return;
}
```

```
main () {
  init_db ();
  read_record (3);
  read_record (11);

}
```

```
$ gcc -o mini-db mini-db.c
$ ./mini-db
ID: 3, Flag: 0
last name:  Mustermann
first name: Thomas
address:    Lothstrasse
town:       Muenchen

ID: 11, Flag: 0
last name:  Mustermann
first name: Thomas
address:    Lothstrasse
town:       Muenchen

$ _
```

File Access in Linux Programs (13)

```
$ hexdump /tmp/mytmpfile
00000000  00 00 00 00 00 00 00 00  4d 75 73 74 65 72 6d 61  |........Musterma|
00000010  6e 6e 00 00 00 00 00 00  1c ef ff bf be 66 07 00  |nn.......ïÿ¿Ÿf..|
00000020  60 ef ff bf 18 75 01 40  f2 82 04 08 20 00 00 00  |`ïÿ¿.u.@ò... ...|
00000030  54 68 6f 6d 61 73 00 00  08 db 04 40 d0 7c 04 40  |Thomas...Û.@Ð|.@|
00000040  a0 7c 01 40 a0 7c 01 40  03 00 00 00 58 7f 01 40  | |.@ |.@....X..@|
00000050  88 7f 01 40 f4 81 04 08  4c 6f 74 68 73 74 72 61  |...@ô...Lothstra|
00000060  73 73 65 00 c4 76 01 40  00 00 00 00 d0 ef ff bf  |sse.Äv.@....Ðïÿ¿|
00000070  9c 82 00 40 f4 6f 01 40  01 00 00 00 18 75 01 40  |...@ôo.@.....u.@|
00000080  70 ef ff bf 56 d0 00 40  c4 76 38 30 30 30 30 4d  |pïÿ¿VÐ.@Äv80000M|
00000090  75 65 6e 63 68 65 6e 00  00 00 00 00 00 00 00 00  |uenchen.........|
000000a0  24 82 04 08 14 9a 04 08  06 00 00 00 f2 82 04 08  |$...........ò...|
000000b0  d0 7c 04 40 f4 1f 15 40  01 00 00 00 00 00 00 00  |Ð|.@ô..@........|
000000c0  4d 75 73 74 65 72 6d 61  6e 6e 00 00 00 00 00 00  |Mustermann......|
000000d0  1c ef ff bf a5 90 7e 00  60 ef ff bf 18 75 01 40  |.ïÿ¿¥.~.`ïÿ¿.u.@|
000000e0  c6 82 04 08 20 00 00 00  54 68 6f 6d 61 73 00 00  |Æ... ...Thomas..|
000000f0  08 db 04 40 70 3a 04 40  a0 7c 01 40 a0 7c 01 40  |.Û.@p:.@ |.@ |.@|
00000100  03 00 00 00 58 7f 01 40  88 7f 01 40 f4 81 04 08  |....X..@...@ô...|
00000110  4c 6f 74 68 73 74 72 61  73 73 65 00 c4 76 01 40  |Lothstrasse.Äv.@|
00000120  00 00 00 00 d0 ef ff bf  9c 82 00 40 f4 6f 01 40  |....Ðïÿ¿...@ôo.@|
00000130  01 00 00 00 18 75 01 40  70 ef ff bf 56 d0 00 40  |.....u.@pïÿ¿VÐ.@|
00000140  c4 76 38 30 30 30 30 4d  75 65 6e 63 68 65 6e 00  |Äv80000Muenchen.|
00000150  00 00 00 00 00 00 00 00  d4 81 04 08 00 9a 04 08  |........Ô.......|
00000160  01 00 00 00 c6 82 04 08  70 3a 04 40 f4 1f 15 40  |....Æ...p:.@ô..@|
00000170  02 00 00 00 00 00 00 00  4d 75 73 74 65 72 6d 61  |........Musterma|
```


9.4.1 Administration of File Systems on Linux

Administration of Linux File Systems (1)

Overview

- create filesystem: **mkfs**, **mkfs.*typ***
- check filesystem consistency: **fsck**, **fsck.*typ***
- mount filesystem: **mount**
- unmount filesystem: **umount**
- special tools: **reiserfstune**, **tune2fs**, **debugfs**
- retrieve information about filesystem: **dumpe2fs**

Administration of Linux File Systems (2)

create a filesystem (= format): **mkfs**, **mkfs.*typ***

- example: ext3 filesystem
  - **mkfs.ext3** or **mke2fs**
  - Syntax: mkfs.ext3 [options] [medium], e.g.
    `mkfs.ext3 -L "Operating Systems" /dev/hda9`

Administration of Linux File Systems (3)

```
# mkfs.ext3 -L "Operating Systems" /dev/hda9
mke2fs 1.36 (05-Feb-2005)
Dateisystem-Label=Operating Systems
OS-Typ: Linux
Blockgröße=1024 (log=0)
Fragmentgröße=1024 (log=0)
2512 Inodes, 10000 Blöcke
500 Blöcke (5.00%) reserviert für den Superuser
erster Datenblock=1
2 Blockgruppen
8192 Blöcke pro Gruppe, 8192 Fragmente pro Gruppe
1256 Inodes pro Gruppe
Superblock-Sicherungskopien gespeichert in den Blöcken:
        8193

Schreibe Inode-Tabellen: erledigt
Erstelle Journal (1024 Blöcke): erledigt
Schreibe Superblöcke und Dateisystem-Accountinginformationen: erledigt

Das Dateisystem wird automatisch alle 28 Mounts bzw. alle 180 Tage überprüft,
je nachdem, was zuerst eintritt. Veränderbar mit tune2fs -c oder -t .
# _
```


Administration of Linux File Systems (4)

Check filesystem consistency:
fsck, **fsck.*typ***

- only use this when the fs is not mounted
- example: ReiserFS file system
  - **fsck.reiserfs** or **reiserfsck**
  - fsck.reiserfs [options] [medium]

```
# fsck.ext3 /dev/hda9
e2fsck 1.36 (05-Feb-2005)
Operating Systems (/dev/hda9): i.O., 11/2512 Dateien,
1366/10000 Blöcke
```


Administration of Linux File Systems (5)

mount filesystem: **mount**

- Syntax:
  mount -t *Type* -o *Options Medium Mount-Point*
  - directory (moint point) must already exist
  - examples:

```
# mount -t ext3 /dev/hda9 /mnt/mnt
# mount -t vfat /dev/fd0 /mnt/floppy
```


Administration of Linux File Systems (6)

- entries in */etc/fstab* define standard mounts

# device	*mount point*	*fs*	*options*	*dump*	*fsck*
/dev/sda6	/	reiserfs	acl,user_xattr	1	1
/dev/sda1	/windows/C	ntfs	ro,users,gid=users,umask=0002,nls=utf8	0	0
/dev/sda3	/windows/D	vfat	users,gid=users,umask=0002,utf8=true	0	0
/dev/sda4	/windows/E	vfat	users,gid=users,umask=0002,utf8=true	0	0
/dev/sda5	swap	swap	pri=42	0	0
devpts	/dev/pts	devpts	mode=0620,gid=5	0	0
proc	/proc	proc	defaults	0	0
usbfs	/proc/bus/usb	usbfs	noauto	0	0
sysfs	/sys	sysfs	noauto	0	0
/dev/dvd	/media/dvd	iso9660	noauto,nosuid,nodev,exec,iocharset=utf8	0	0
/dev/dvdrecorder	/media/dvdrecorder	iso9660	noauto,nosuid,nodev,exec,iocharset=utf8	0	0

- mounting such filesystems without options:

```
# mount /media/dvd
# mount /windows/D
```

Administration of Linux File Systems (7)

unmount filesystem: **umount**

- **umount** accepts either the device name or the mount point;
- **umount** finds information about mounted filesystems in */etc/mtab*

```
# umount /media/dvd
# umount /windows/D
```


Administration of Linux File Systems (8)

special tools: **reiserfstune**, **tune2fs**, **debugfs**

```
NAME
       reiserfstune - The tuning tool for the ReiserFS filesystem.

SYNOPSIS
       reiserfstune  [ -f ] [ -j | --journal-device FILE ] [ --no-journal-available ] [
       --journal-new-device FILE ] [ --make-journal-standard ] [  -s  |  --journal-new-
       size  N ] [ -o | --journal-new-offset N ] [ -t | --max-transaction-size N ] [ -b
       | --add-badblocks file ] [ -B | --badblocks file ] [ -u | --uuid UUID ] [  -l  |
       --label LABEL ] device

DESCRIPTION
       reiserfstune  is used for tuning the ReiserFS. It can change two journal parame-
       ters (the journal size and the maximum transaction size), and it  can  move  the
       journal's  location to a new specified block device. (The old ReiserFS's journal
       may be kept unused, or discarded at the user's option.) Besides  that  reiserfs-
       tune can store the bad block list to the ReiserFS and set UUID and LABEL.  Note:
       At the time of writing the relocated  journal  was  implemented  for  a  special
       release  of  ReiserFS, and was not expected to be put into the mainstream kernel
       until approximately Linux 2.5.  This means that if you have the stock kernel you
       must  apply  a special patch. Without this patch the kernel will refuse to mount
       the newly modified file system.  We will charge $25 to explain this  to  you  if
       you ask us why it doesn't work.

       Perhaps the most interesting application of this code is to put the journal on a
       solid state disk.
```


Administration of Linux File Systems (9)

```
NAME
       tune2fs - adjust tunable filesystem parameters on ext2/ext3 filesystems

SYNOPSIS
       tune2fs [ -l ] [ -c max-mount-counts ] [ -e errors-behavior ] [ -f ] [ -i inter-
       val-between-checks ] [ -j ] [ -J journal-options ] [ -m reserved-blocks-percent-
       age  ]  [ -o [^]mount-options[,...]  ] [ -r reserved-blocks-count ] [ -s sparse-
       super-flag ] [ -u user ] [ -g group ] [ -C mount-count ] [ -L volume-name ] [ -M
       last-mounted-directory  ] [ -O [^]feature[,...]  ] [ -T time-last-checked ] [ -U
       UUID ] device

DESCRIPTION
       tune2fs allows the system administrator to  adjust  various  tunable  filesystem
       parameters on Linux ext2/ext3 filesystems.
```

```
NAME
       debugfs - ext2/ext3 file system debugger

SYNOPSIS
       debugfs  [  -Vwci  ]  [  -b  blocksize  ] [ -s superblock ] [ -f cmd_file ] [ -R
       request ] [ -d data_source_device ] [ device ]

DESCRIPTION
       The debugfs program is an interactive file system debugger. It can  be  used  to
       examine and change the state of an ext2 file system.
       device  is the special file corresponding to the device containing the ext2 file
       system (e.g /dev/hdXX).
```


Administration of Linux File Systems (10)

query information: **dumpe2fs**

```
# dumpe2fs /dev/hda9
dumpe2fs 1.36 (05-Feb-2005)
Filesystem volume name:   Betriebssysteme
Last mounted on:          <not available>
Filesystem UUID:          2127b3b7-0a24-441f-83c5-30ea11641d85
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              2512
Block count:              10000
Reserved block count:     500
Free blocks:              8634
Free inodes:              2501
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         1256
Inode blocks per group:   157
Filesystem created:       Mon Dec 11 23:59:50 2006
Last mount time:          Tue Dec 12 00:06:37 2006
Last write time:          Tue Dec 12 00:06:37 2006
Mount count:              1
Maximum mount count:      28
Last checked:             Mon Dec 11 23:59:50 2006
Check interval:           15552000 (6 months)
...
```

Administration of Linux File Systems (11)

- which filesystems does the running Linux kernel support? -> look at **/proc/filesystems**
- Linux loads file system drivers (as modules) dynamically, when a **mount** command for a (not yet) supported file system is executed

```
$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   debugfs
nodev   pipefs
nodev   futexfs
nodev   tmpfs
nodev   eventpollfs
nodev   devpts
        ext2
nodev   ramfs
nodev   hugetlbfs
        minix
        iso9660
nodev   nfs
nodev   mqueue
nodev   rpc_pipefs
        reiserfs
nodev   usbfs
        ntfs
        vfat
nodev   subfs
        hfsplus
```