Materialsammlung Betriebssysteme und betriebssystemnahe Programierung

Prof. Dr. Hans-Jürgen Buhl

Sommersemester 2005

Bergische Universität Wuppertal
Fachbereich C — Mathematik

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Dokumentation zu System Calls . . . 7
1.2 API-Dokumentation . . . 13

2 Betriebssysteme - eine Übersicht **17**
2.1 Aufgaben eines Betriebssystems . . . 17
2.2 Modularer Aufbau vom Windows 2000 . . . 19
2.2.1 HAL = Hardware Abstraction Layer . . . 19
2.2.2 **Der Kern des Windows-Kernels** . . . 22
2.2.3 **Kernel-Module** . . . 22
2.2.4 **Systemdienste und Systemschnittstelle** . . . 24
2.2.5 **Subsysteme** . . . 24
2.3 System-Architektur von UNIX/Linux . . . 25
2.4 Virtuelle Betriebssysteme . . . 27
2.5 Secure global desktop/Virtual network computing . . . 27
2.6 Linken und Laden: statische und shared Bibliotheken . . . 28
2.6.1 Linken . . . 28
2.6.2 Compilationsphasen . . . 33
2.6.3 Statisches und dynamisches Linken im Vergleich . . . 35
2.6.4 Generierung einer statischen Bibliothek libmylib1.a . . . 35
2.6.5 Generierung einer shared Bibliothek libmylib1.so.0.0 . . . 35
2.6.6 Memory-Layout von Prozessen . . . 36
2.7 Lowlevel-Aufbau von Festplatten . . . 40
2.7.1 Physikalischer Aufbau . . . 40
2.7.2 Die magnetische Unterteilung der Festplatte . . . 44

3 Partitionen und Dateisysteme **45**
3.1 Virtuelle Festplatten . . . 45
3.1.1 Zweck von Festplatten-Partitionierung . . . 45
3.1.2 Partitionieren unter UNIX/SOLARIS . . . 46
3.1.3 Windows-Partitionen . . . 48
3.2 Dateisysteme erzeugen . . . 53
3.2.1 Dateisystem-Erzeugung unter Windows . . . 56
3.2.2 Dateisystemerzeugung unter Linux . . . 56
3.2.3 Anlegen eines UNIX-Dateisystems unter Solaris . . . 57
3.2.4 Zusammenfügen von UNIX-Paritionen: die Mount-Tabelle . . . 58

3.2.5 MultiBoot-System: Windows-Partitionen im Boot-Chooser 58
3.3 Dateisystem-Übersicht 59
3.3.1 Ältere Betriebssysteme 59
3.3.2 Neuere Betriebssysteme 59
3.4 Access Control Lists — ACLs 61
3.5 Namensräume im Netzwerk / Verteiltes Rechnen 61
3.6 Verteilte Betriebssysteme 62
3.7 Das FAT-Dateisystem 63
3.7.1 Aufbau eines FAT-Dateisystems 63
3.7.2 FAT-Verzeichnisse 64
3.8 Das UNIX Filesystem UFS/UFS2 65
3.8.1 Inodes 67
3.9 Das WINDOWS-Dateisystem NTFS 69
3.9.1 NTFS Schlüssel-Designkonzepte 70
3.9.2 Die NTFS Volumen- und Dateistruktur 71
3.9.3 Recoverability/Wiederherstellbarkeit 71
3.10 Weitere UNIX- und LINUX-Dateisysteme 73
3.10.1 S5FS 73
3.10.2 FFS oder UFS 73
3.10.3 LFS 73
3.10.4 ext2 und ext3 73
3.10.5 XFS 74
3.10.6 JFS 74
3.10.7 Reiserfs 74
3.10.8 HFS+ 74
3.10.9 ZFS 75

4 RAID **77**
4.1 RAID 0 77
4.2 RAID 1 78
4.3 RAID 5 78
4.4 RAID 6 79
4.5 Kombinations-RAIDs 80

5 Parallel arbeitende Applikationen **81**
5.1 Das Klonen von Prozessen 81
5.2 Starten von (detachten) Threads 81

6 Virtueller Speicher **83**
6.1 Segmentierung 83
6.2 Paging 83
6.3 Virtueller Speicher 83

7 Sicherheit beim Programmieren **85**
7.1 Sicherheitsaspekte 85
7.2 DFN-CERT 87
7.2.1 Puffer-Überläufe und Formatstring-Schwachstellen 87
7.3 Präventive Schutzmöglichkeiten für Programmierer 88
7.4 Linux gegen Buffer-Overflows schützen 89
7.5 Buffer-Overflow-Schutz bei Windows XP 89
7.6 Secure Linux 89

Abbildungsverzeichnis

1.1 sun.doc.com . . . 7
1.2 Suche **dup2**-Dokumentation . . . 8
1.3 Suchresultat: **dup2** . . . 8
1.4 **dup2** im Manual-Abschnitt 2 . . . 8
1.5 **dup2**-Dokumentation . . . 9
1.6 Auszug aus dem INDEX für System Calls . . . 10
1.7 Das Manual in der Konsole: **man** . . . 11
1.8 Dokumentation für **date** . . . 11
1.9 annähernde Suche . . . 12
1.10 Manual-Seiten in kde . . . 12
1.11 man pages section 3 — die API-Dokumentation: . . . 13
1.12 Baisc Libary Funktions . . . 13
1.13 passwd, printf, ... 14
1.14 printf-Dokumentation . . . 15

2.1 Windows 2000 Systemarchitektur . . . 19
2.2 HAL-Varianten . . . 21
2.3 Linux-Architektur . . . 25
2.4 Solaris2-Architektur . . . 25
2.5 Solaris Kernel Overview . . . 26
2.6 Spuren und Sektoren einer Festplatte . . . 43

3.1 Partitionieren einer Solaris-Festplatte mittels **format** . . . 46
3.2 Partitionieren einer Solaris-Festplatte mittels **format** (Fortsetzung) . . . 47
3.3 Vier primäre Partitionen . . . 48
3.4 Partitionierung unter Windows . . . 56
3.5 Partitionierung unter Linux . . . 57
3.6 Anlegen einer Partition mit **newfs** . . . 57
3.7 Mount-Tabelle . . . 58
3.8 Windows Partitionen . . . 58
3.9 Windows Shares . . . 61
3.10 Verteilte Betriebssysteme . . . 62
3.11 Struktur und Aufbau einer FAT-Partition . . . 63
3.12 Aufbau einer UFS-Partition . . . 66
3.13 Adressenschema für die Datenblöcke eines Inodes . . . 68
3.14 NTFS-Partition . . . 69

3.15 NTFS-Komponenten 71

4.1 RAID 0 77
4.2 RAID 1 78
4.3 RAID 5 78
4.4 RAID 6 79
4.5 Kombinations-RAID 10 80

7.1 Internetseite http://www.cert.dfn.de 87
7.2 Secure UNIX Programming FAQ, ... 89

Tabellenverzeichnis

2.1 Windows 2000 Kernel-Module 23

3.1 Datenträgertypen 50
3.2 Partitionstabellenfelder 51
3.3 File und Datei Attribute 64
3.4 UFS Datei- und Dateisystemgrößen 65
3.5 Inode-Aufbau 67
3.6 NTFS File- und Verzeichnis-Attribut-Typen 70

Einordnung der Vorlesung / Vorbemerkung

Vorlesung/Einordnung: Hauptstudium Diplom/Nebenfach Informatik: Praktische und technische Informatik (zusammen mit einer weiteren 2SWS-Veranstaltung); Master Wirtschaftsmathematik: Wahlpflichtveranstaltung Informatik; Bachelor IT: Wahlpflichtmodul Praktische Informatik B; Diplom/Bachelor/Master Wirtschaftswissenschaft Hauptstudium: Wahlpflichtfach Methoden der Angewannten und Praktischen Informatik: Modul IV - Betriebssysteme; Nebenfächer und Studienschwerpunkte Informatik anderer Studiengänge.

Vorkenntnisse: Erfolgreiche Teilnahme am CM.500; Grundzüge der Informatik.

Inhalt: Nach einer grundlegenden Einfhrung in Aufgabe und Aufbau eines Betriebssystems werden exemplarisch Dienste und Schnittstellen bis hin zur programmtechnischen Integration mit Anwendungsprogrammen vorgestellt.

Literatur: wird in der Vorlesung bekannt gegeben.

Übungen zu Bertriebssysteme - Konzepte, Dienste, Schnittstellen: Buhl

1 Einleitung

System Calls anstatt reinen Hochsprachencodes braucht man wegen:

- Vermeidung von Overhead (schnellerer Code)
- Benutzung von Diensten, die eine Hochsprache nicht bietet (File- und Record-Locking, Interprozeßkommunikation, etc.)

Ein Beispiel: test-read.c

```
#include <stdio.h>

int main(void)
{
  char buf[1024];
  int len;

  if ((len = read(5, buf,sizeof(buf))) == -1)
    perror("read");

  return 0;
}
```

liefert beim Programmaufruf:

```
./test-read
read: Bad file descriptor
```

Beachte: Bei jedem SystemCall muß der Erfolgsstatus überpfüft und im folgenden Programmablauf berücksichtigt werden!

Einige weitere Beispiele:

lock.c

```
/* lock.c -- simple example of record locking */

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

/* displays the message, and waits for the user to press
   return */
void waitforuser(char * message) {
    char buf[10];

    printf("%s", message);
    fflush(stdout);

    fgets(buf, 9, stdin);
}

/* Gets a lock of the indicated type on the fd which is passed.
   The type should be either F_UNLCK, F_RDLCK, or F_WRLCK */
void getlock(int fd, int type) {
    struct flock lockinfo;
    char message[80];

    /* we'll lock the entire file */
    lockinfo.l_whence = SEEK_SET;
    lockinfo.l_start = 0;
    lockinfo.l_len = 0;

    /* keep trying until we succeed */
    while (1) {
        lockinfo.l_type = type;
        /* if we get the lock, return immediately */
        if (!fcntl(fd, F_SETLK, &lockinfo)) return;

        /* find out who holds the conflicting lock */
        fcntl(fd, F_GETLK, &lockinfo);

        /* there's a chance the lock was freed between the F_SETLK
           and F_GETLK; make sure there's still a conflict before
```

```
            complaining about it */
        if (lockinfo.l_type != F_UNLCK) {
            sprintf(message, "conflict with process %d... press "
                    "<return> to retry:", lockinfo.l_pid);
            waitforuser(message);
        }
    }
}

int main(void) {
    int fd;

    /* set up a file to lock */
    fd = open("testlockfile", O_RDWR | O_CREAT, 0666);
    if (fd < 0) {
        perror("open");
        return 1;
    }

    printf("getting read lock\n");
    getlock(fd, F_RDLCK);
    printf("got read lock\n");

    waitforuser("\npress <return> to continue:");

    printf("releasing lock\n");
    getlock(fd, F_UNLCK);

    printf("getting write lock\n");
    getlock(fd, F_WRLCK);
    printf("got write lock\n");

    waitforuser("\npress <return> to exit:");

    /* locks are released when the file is closed */

    return 0;
}
```

daytime.c

```
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

int main() {
   struct timeval tv;
   struct timezone tz;
   time_t now;
   /* beginning_of_time is smallest time_t-sized value */
   time_t beginning_of_time = 1L<<(sizeof(time_t)*8 - 1);
   /* end_of_time is largest time_t-sized value */
   time_t end_of_time = ~beginning_of_time;

   printf("time_t is %d bits long\n\n", sizeof(time_t)*8);

   gettimeofday(&tv, &tz);
   now = tv.tv_sec;
   printf("Current time of day represented as a struct timeval:\n"
          "tv.tv_sec = 0x%08x, tv.tv_usec = 0x%08x\n"
          "tz.tz_minuteswest = 0x%08x, tz.tz_dsttime = 0x%08x\n\n",
          tv.tv_sec, tv.tv_usec, tz.tz_minuteswest, tz.tz_dsttime);

   printf("Demonstrating ctime()%s:\n", sizeof(time_t)*8 <= 32 ? "" :
          " (may hang after printing first line; press control-C)");
   printf("time is now %s", ctime(&now));
   printf("time begins %s", ctime(&beginning_of_time));
   printf("time ends %s", ctime(&end_of_time));

   exit (0);
}
```

cat1.c

```
/* cat1.c -- simple version of cat */
#include <stdio.h>
#include <unistd.h>

/* While there is data on standard in (fd 0), copy it to standard
   out (fd 1). Exit once no more data is available. */

int main(void) {
    char buf[1024];
    int len;

    /* len will be >= 0 while data is available, and read() is
       successful */
    while ((len = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
        if (write(STDOUT_FILENO, buf, len) != len) {
            perror("write");
            return 1;
        }
    }

    /* len was <= 0; If len = 0, no more data is available.
       Otherwise, an error occurred. */
    if (len < 0) {
        perror("read");
        return 1;
    }

    return 0;
}
```

mysleep.c

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int usecsleep(int secs, int usecs) {
    struct timeval tv;

    tv.tv_sec = secs;
    tv.tv_usec = usecs;

    return select(0, NULL, NULL, NULL, &tv);
}

int main(){
    usecsleep(0, 500000);
}
```

Aufgabe 1.1 *Bringen Sie die Beispiele auf Ihrem Computer zum Ablauf und versuchen Sie die Quellcodes Zeile für Zeile zu verstehen, indem Sie die Dokumentation gemäß Abschnitt 1.1 zu Rate ziehen.*

1.1 Dokumentation zu System Calls und zur (C)-Laufzeitbibliothek

Auf den Web-Seiten der Firma **SUN** kann man unter http://docs.sun.com/app/docs/prod/3253 nach einzelnen System Calls und deren Bedeutung suchen und sich die entsprechende Dokumentation anschauen:

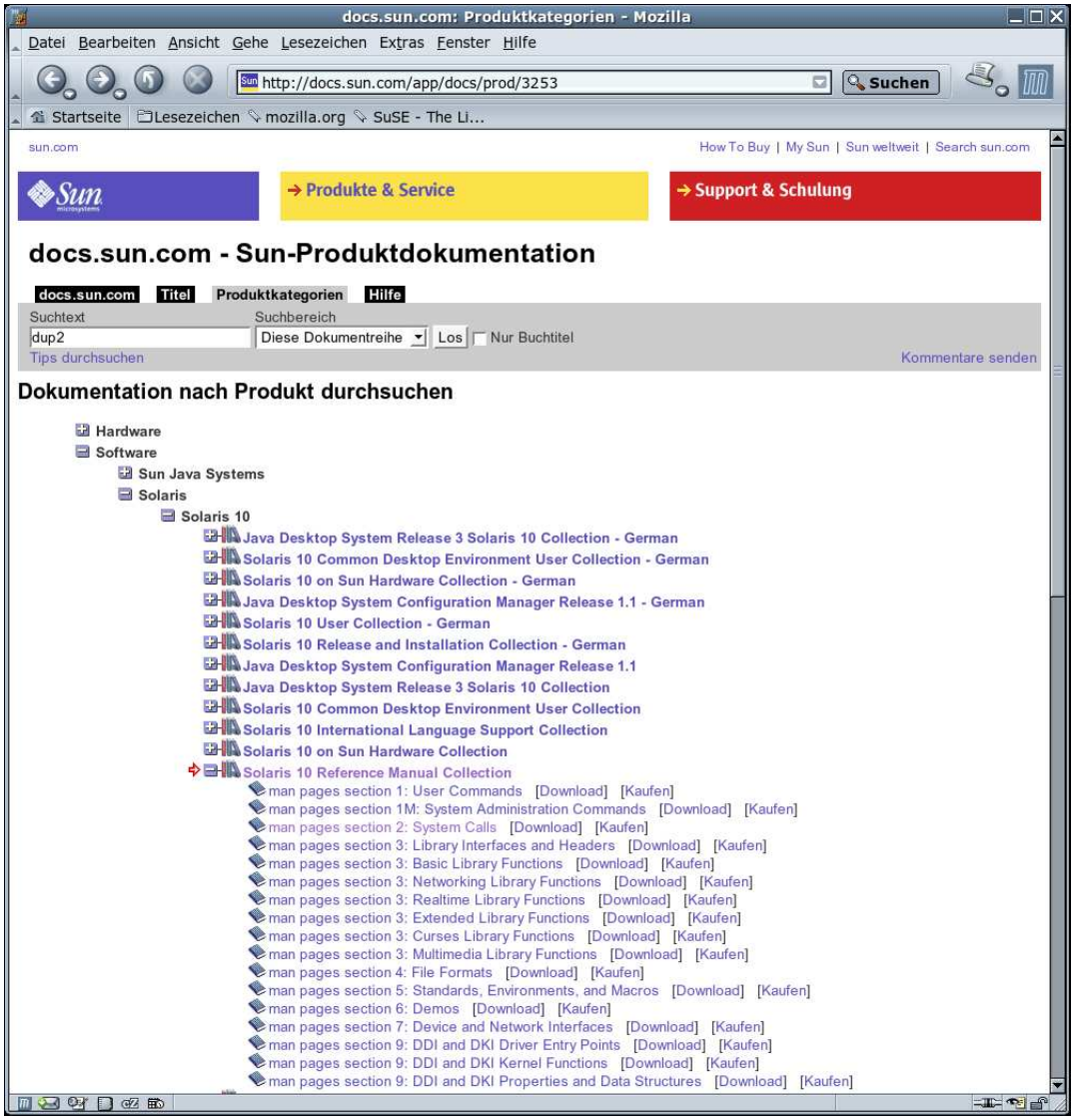


Abbildung 1.1: sun.doc.com

Als Beispiel suchen wir in den man-Pages (Manual-Seiten) nach dem System Call: **dup2**

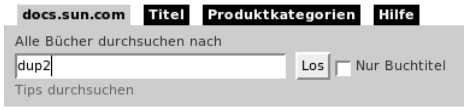


Abbildung 1.2: Suche **dup2**-Dokumentation

Das Suchresultat liefert:

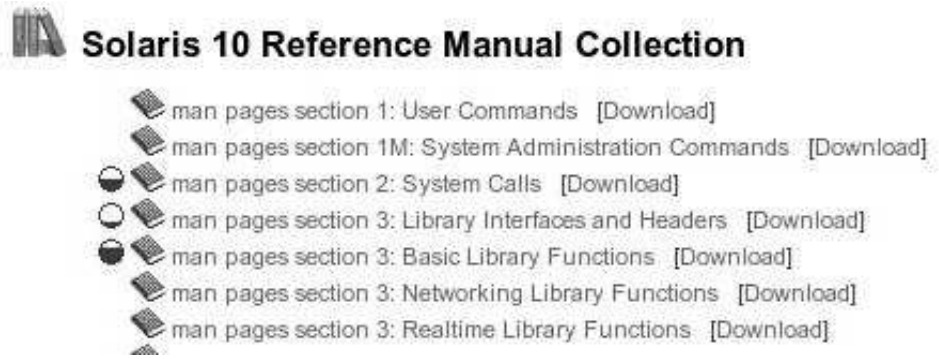


Abbildung 1.3: Suchresultat: **dup2**

In den **man pages section 2: System Calls** kann man sich die gewünschten Erläuterungen anzeigenlassen lassen:

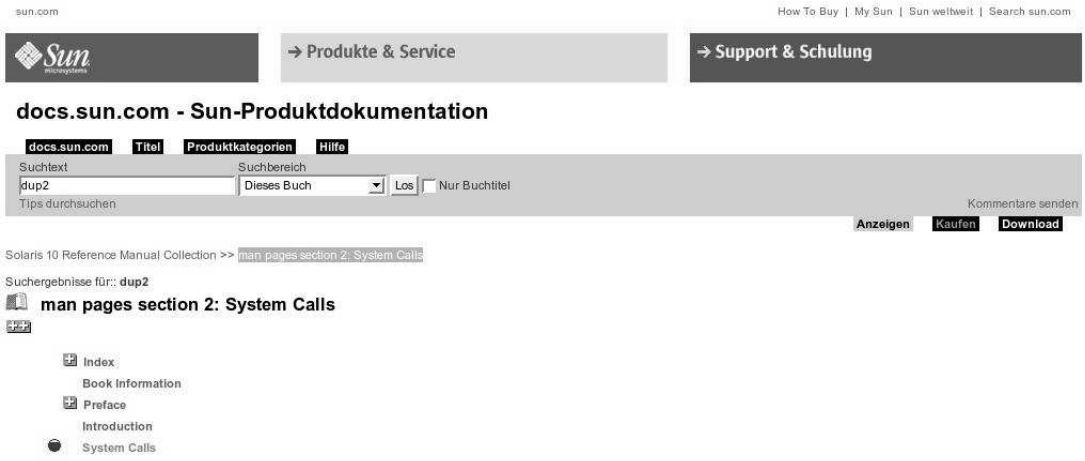


Abbildung 1.4: **dup2** im Manual-Abschnitt 2

dup2 hat folgenden Eintrag:

dup(2)

NAME

dup– duplicate an open file descriptor

SYNOPSIS

```
#include <unistd.h>
```

int dup(int *fildes*);

DESCRIPTION

The **dup()** function returns a new file descriptor having the following in common with the original open file descriptor *fildes*:

- same open file (or pipe)
- same file pointer (that is, both file descriptors share one file pointer)
- same access mode (read, write or read/write).

The new file descriptor is set to remain open across *exec* functions (see fcntl(2)).

The file descriptor returned is the lowest one available.

The `dup(fildes)` function call is equivalent to:

```
fcntl(fildes, F_DUPFD, 0)
```

RETURN VALUES

Upon successful completion, a non-negative integer representing the file descriptor is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS

The **dup()** function will fail if:

`EBADF`

The *fildes* argument is not a valid open file descriptor.

`EINTR`

A signal was caught during the execution of the **dup()** function.

`EMFILE`

The process has too many open files (see getrlimit(2)).

`ENOLINK`

The *fildes* argument is on a remote machine and the link to that machine is no longer active.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SEE ALSO

close(2), creat(2), exec(2), fcntl(2), getrlimit(2), open(2), pipe(2), dup2(3C), lockf(3C), attributes(5)

SunOS 5.9 Last Revised 28 Dec 1996

Abbildung 1.5: **dup2**-Dokumentation

Der **INDEX** der Section 2 liefert eine sehr schöne Übersicht über die System Calls des UNIX-Betriebssystems.

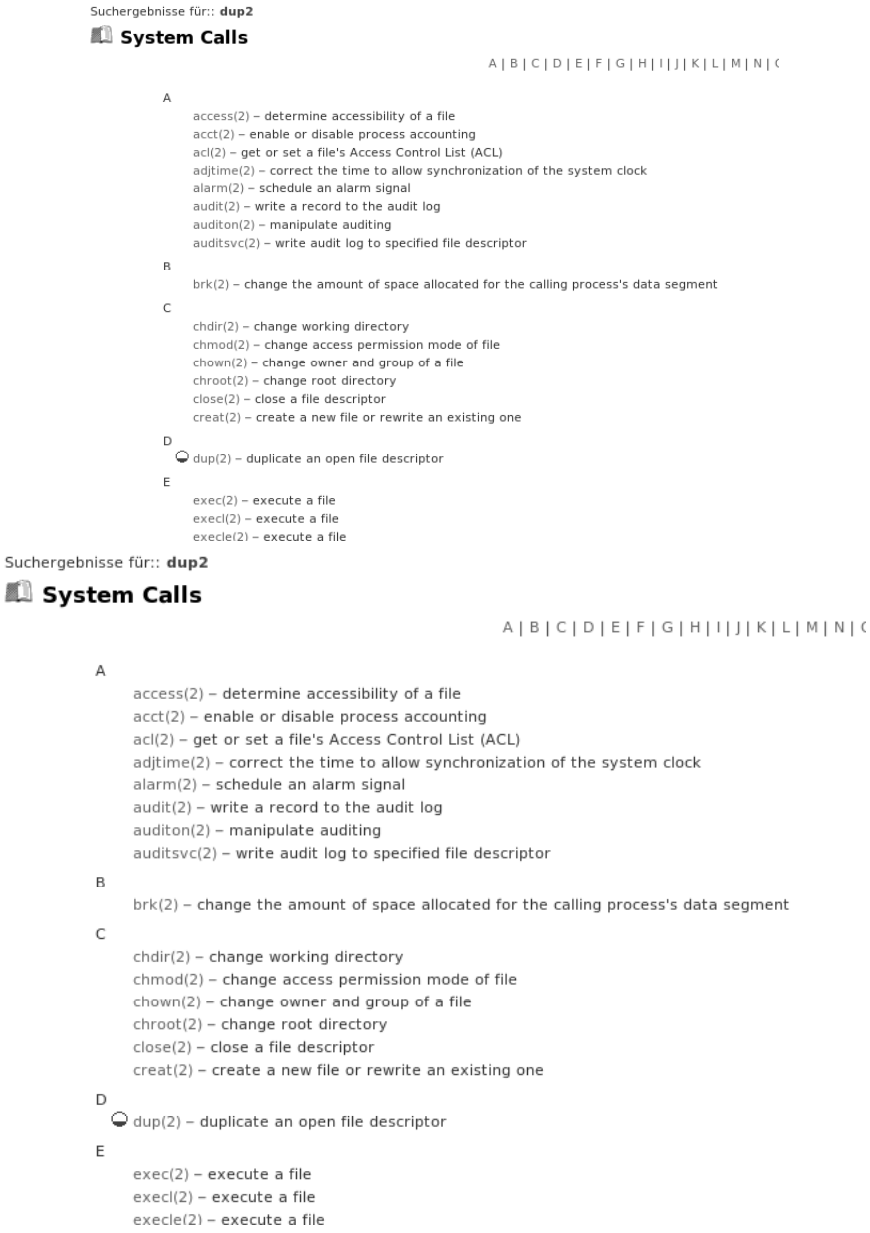


Abbildung 1.6: Auszug aus dem INDEX für System Calls

Aufgabe 1.2 *Informieren Sie sich über den SystemCall* `brk(0)`*, der beim Start eines Programms vom Loader (für welchen Zweck?) benutzt wird.*

Eine weitere Dokumentationsquelle für SystemCalls sind die lokalen `man-pages`. Der Aufruf erfolgt in einer Konsole mit dem Kommando `man`. Mittels `man man` erhält man zum Beispiel die Manualseite des Befehls `man` (für Manual).

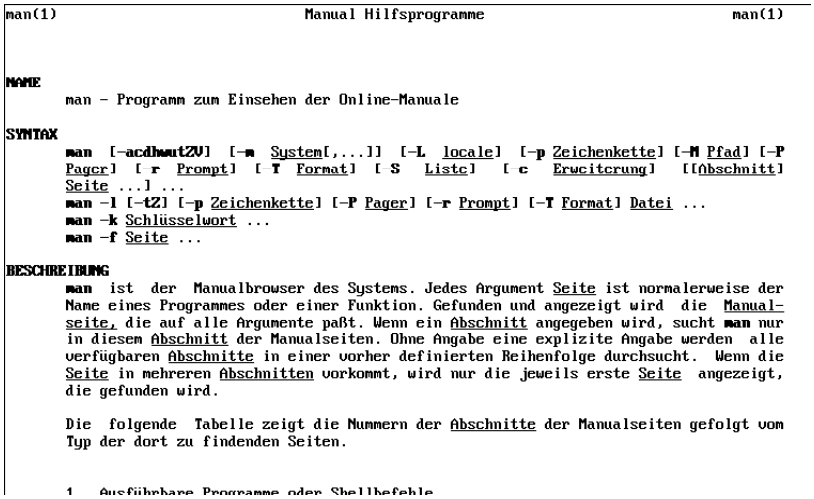


Abbildung 1.7: Das Manual in der Konsole: **man**

Nach Informationen des Kommandos `date` sucht man also mittels `man date`:

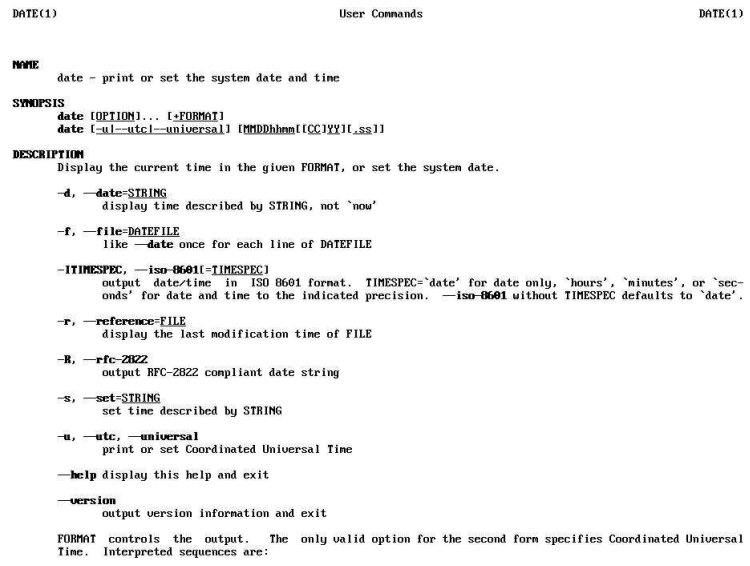


Abbildung 1.8: Dokumentation für **date**

Bei Unkenntnis des genauen Kommandos für einen gesuchten Zweck, kann man mittels `apropos` einen Index durchsuchen:

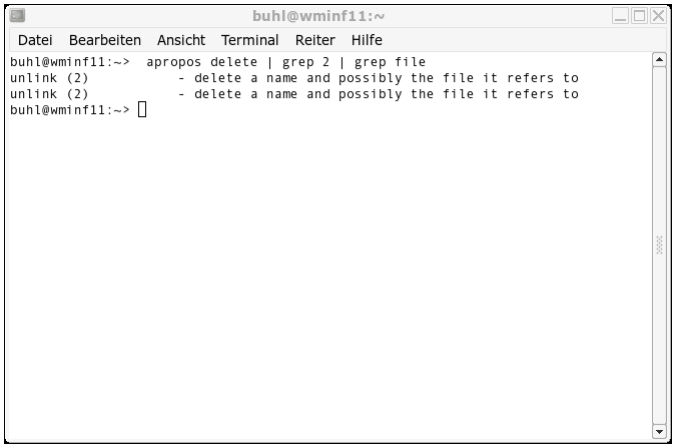


Abbildung 1.9: annähernde Suche

Schließlich kann man auch `kdehelp` (bezeihungsweise `susehelp`) zur bequemen Suche nach Manual-Informationen benutzen:

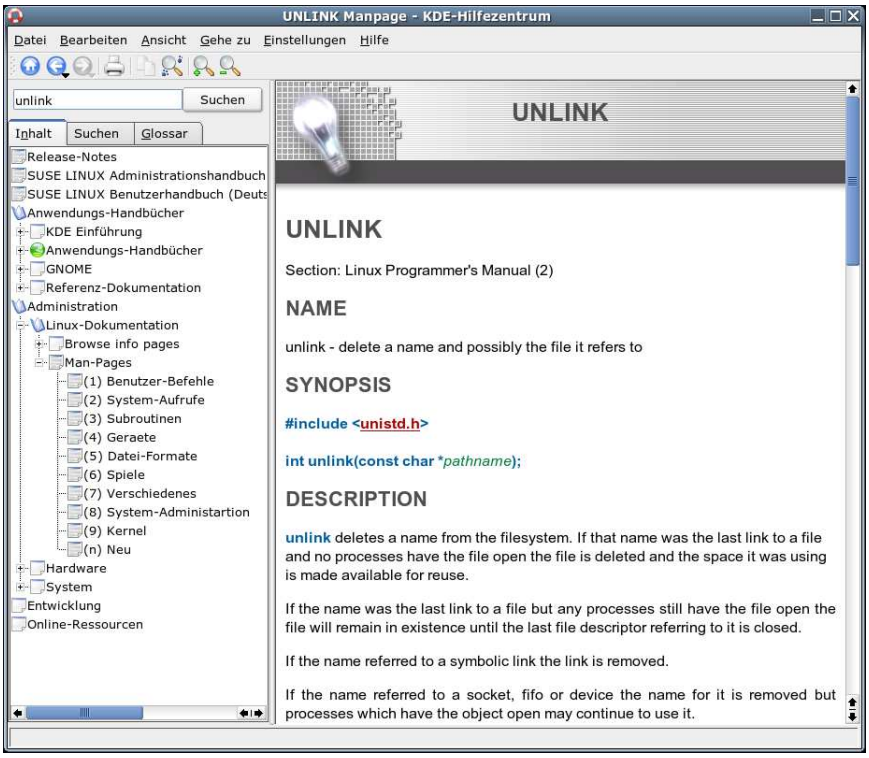


Abbildung 1.10: Manual-Seiten in kde

1.2 API-Dokumentation

Die **man pages section 3** enthalten die **API-Dokumentation** der Laufzeitbibliotheken:

Abbildung 1.11: man pages section 3 — die API-Dokumentation:

Beispiele:

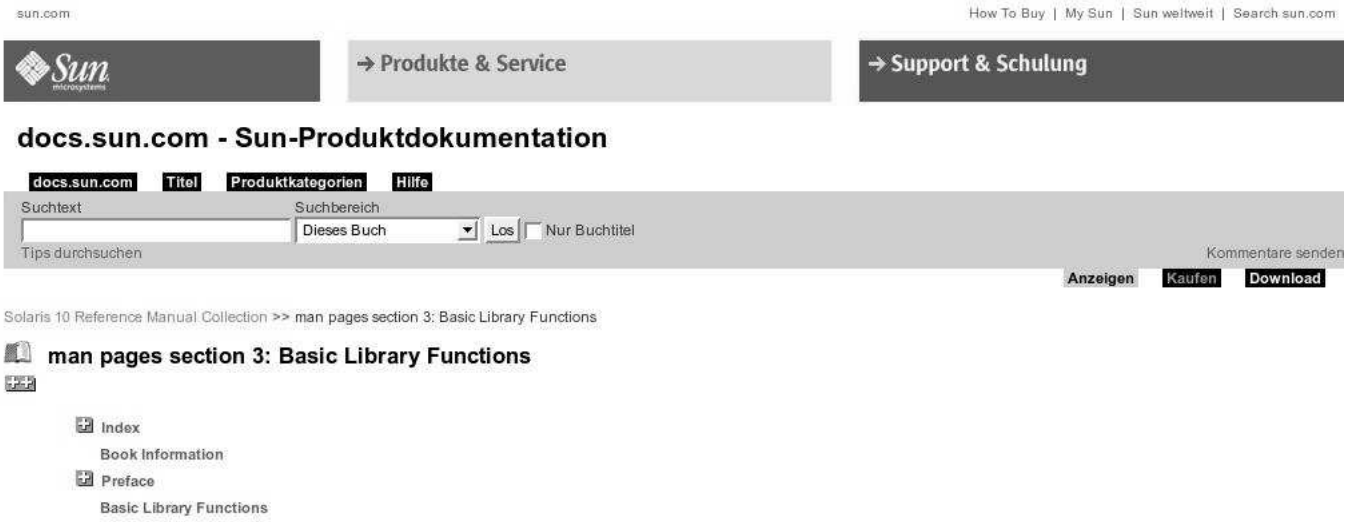


Abbildung 1.12: Baisc Libary Funktions

Der INDEX unter P zeigt beispielsweise **psignal**, **printf**, **pthread** ...:

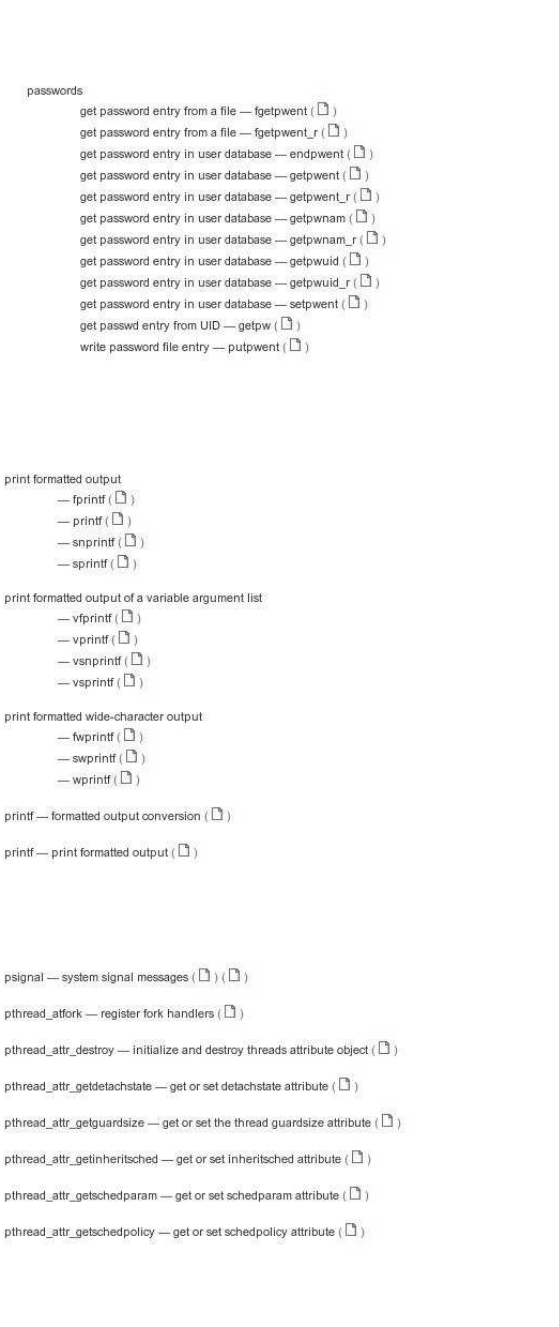


Abbildung 1.13: passwd, printf, ...

Die Dokumentation zu **printf** beginnt folgendermaßen:

printf(3C)

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUES | ERRORS | USAGE | EXAMPLES | ATTRIBUTES | SEE ALSO | NOTES

NAME

printf, fprintf, sprintf, snprintf– print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf(const char *restrict format, /* args*/ ...);
int fprintf(FILE *restrict stream, const char *restrict format, /* args*/ ...);
int sprintf(char *restrict s, const char *restrict format, /* args*/ ...);
int snprintf(char *restrict s, size_t n, const char *restrict format, /* args*/ ...);
```

DESCRIPTION

The **printf**() function places output on the standard output stream `stdout`.

The **fprintf**() function places output on on the named output stream *stream*.

The **sprintf**() function places output, followed by the null byte (`\0`), in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available.

The **snprintf**() function is identical to **sprintf**() with the addition of the argument *n*, which specifies the size of the buffer referred to by *s*. If *n* is 0, nothing is written and *s* can be a null pointer. Otherwise, output bytes beyond the *n*-1st are discarded instead of being written to the array and a null byte is written at the end of the bytes actually written into the array.

⋮

Abbildung 1.14: printf-Dokumentation

Aufgabe 1.3 *Lesen Sie die Dokumentation zu* `getchar()`, `getch()` *sowie zu* `cbreak()`. *Stellen Sie kurz die Möglichkeiten dieser Funktionen dar.*

Weitere Informationsquellen:

- http://www.schellong.de/c.htm
- http://www.gnu.org/software/libc/manual

Aufgabe 1.4 *Informieren Sie sich mittels der weiteren Informationsquellen über die POSIX-Funktionen und klassifizieren Sie diese nach dem Verwendungszweck.*

2 Betriebssysteme - eine Übersicht

2.1 Aufgaben eines Betriebssystems

- Ablaufsteuerung

  Starten und Beenden von Programmen/Prozessen, Abbrechen von Programmen (kill), ...

- Abstraktion

  – techn. Einzelheiten vor Benutzern und Anwendungsprogrammen verstecken (z.B. HAL für Geräteschnittstellen, Chipsatz, usw.)

  Vergleiche dazu: http://support.microsoft.com/kb/q148659/#XSLTH3156121122120121120120

- APIs (und ABIs) bereitstellen für

  – Anwendungsprogramme (z.B. den System Calls, C-Laufzeitbibliotheken,...)

  – Compilerhersteller (C-, C++-Calling-Konventionen, Word-Alignment in Structures, Speicherlayout in Unions, Register- und Stack-Nutzung zur Parameterübergabe bei Funktionsaufrufen, Name-Mangling bei überladenen Funktionen, ...)

  Vergleiche: ABI LP64 in http://www.unix.org/version2/whatsnew/lp64_wp.html,
  Sun's C++ ABI in http://developers.sun.com/tools/cc/articles/CC_abi/CC_abi_content.html

- Koordinierung und Zuteilung von Betriebsmittel, (Multitasking)

  – Prozessor(en)

  – Hauptspeicher

  – Hintergrundspeicher

  – Zugriffsrechte auf E/A-Geräte, Dateisysteme, ...

  – Initialisierung von Gerätetreibern

  – Auftragsverwaltung (Jobs, Cronjobs, ...)

  – Ausnahmebedinungsbehandlung, Interrupt-Tabellen initialisieren

  – ...

  Vergeiche Interrupt-Handler in:
  http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=interrupt+handler&action=Search

- Schutz mehrerer nacheinander oder gleichzeitig arbeitenden Nutzer bzw. Prozesse voreinander (Multitasking/Multiuser)
- Schutz des Betriebssystems vor den Nutzern / Anwendungsprogrammen
- Bedienschnittstellen
  - Shells, awk, sed, Perl, Python, ...
  - GUIs
  - ...

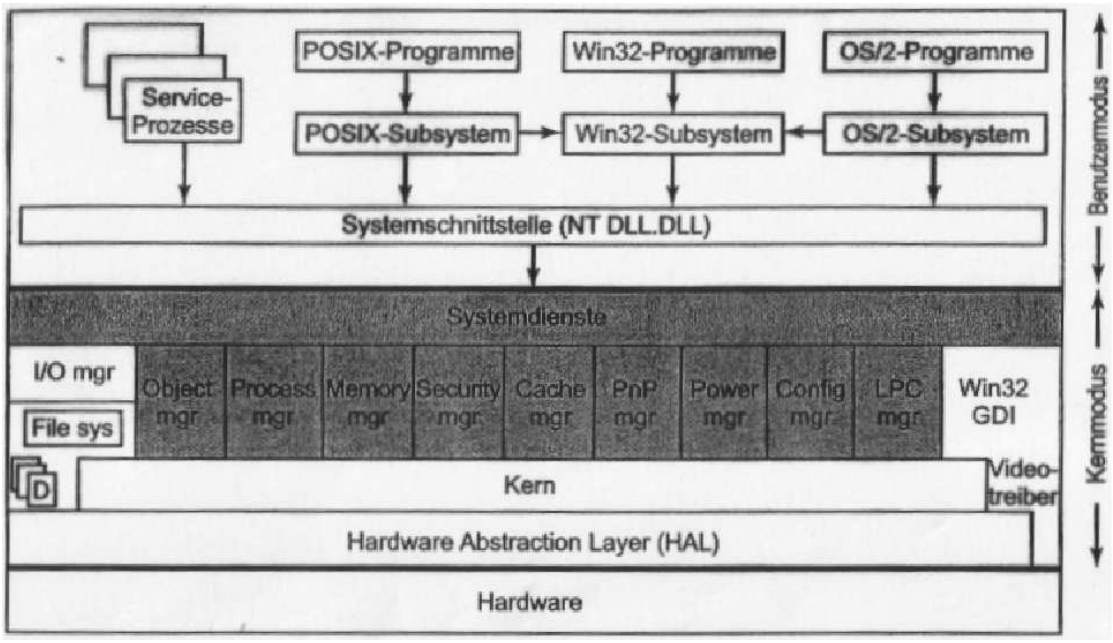


Abbildung 2.1: Windows 2000 Systemarchitektur

2.2 Modularer Aufbau vom Windows 2000 - Schichtweiser Aufbau (Layers)

Windows ist in Schichten aufgebaut, wobei die unteren (maschinennäheren) Schichten die Basis für die oberen (maschinenferneren) Schichten bilden. Die oberen Schichten können die Funktionen der unteren Schichten aufrufen, aber nicht umgekehrt.

2.2.1 HAL = Hardware Abstraction Layer (Hardwareabstraktions-Schicht)

- Chipsatz- und Motherbord-abhängige Unterschiede verstecken
- einheitlich Dienste bereitstellen:
  - Schnittstellen zu BIOS und BIOS-Setup
  - Abbildung **busbezogener Geräteadressen** auf **logische Adressen** (Beispiel: **con, aux, prn, com1, lpt1** in MSDOS)
  - nur HAL greift auf Geräteregister zu
  - Interruptroutinen setzen/Prioritäten festlegen
  - Vereinheitlichung dem Rest des OS gegenüber:
    - Systembus

        * DMA-Controller
        * Interrupt-Controller
        * Memmory Module
        * System-Timer (Einheit: 0,1 ms)
    - SMP (Symmetric Multiprocessing)

  Benutzt werden standartisierte API's wie z.B.: Win32, GDI, MDI, ...

Beispiele für Hardware-Abstraktionen sind:

- Druckertreiber
- Grafikschnittstellen (z.B. OpenGL oder DirectX)
- virtuelle Maschinen (VM)

DirectX umgeht HAL, um schnelle Multimedia-Prozeduren durch direkten Hardwarezugriff zu ermöglichen.

HAL ist für einen Großteil der Anwendungsprogramme und viele Systemprozesse notwendig. Wenige Bestandteile des Kernels und die Gerätetreiber kommunizieren direkt mit der Hardware, ohne HAL zu nutzen.

Auf einer Windows2000-System-CD liegen mehrere HAL-Versionen bereit, die je nach Hardware als

```
\WINNT\SYSTEM32\HAL.DLL
```

installiert werden.

In Windows NT gab es sogar Versionen für Digital Equipment Corporation (DEC) RISC-Workstations:

```
HAL NAME          COMPUTER MODEL NUMBER
--------          ---------------------

halmikas.dll = "Digital AlphaServer 1000 Family Uniprocessor"
hal0jens.dll = "Digital DECpc AXP 150"
halsabmp.dll = "Digital AlphaServer 2x00 4/xxx Family"
halsabmp.dll = "Digital AlphaServer 2100A 4/xxx"
halavant.dll = "Digital AlphaStation 200/400 Family"
halnonme.dll = "Digital AXPpci 33"
halqs.dll    = "Digital Multia MultiClient Desktop"
halalcor.dll = "Digital AlphaStation 500/600 Family"
halgammp.dll = "Digital AlphaServer 2x00 5/xxx Family"
halgammp.dll = "Digital AlphaServer 2100A 5/xxx"
haleb64p.dll = "Digital AlphaPC64"
haleb164.dll = "Digital Alpha EB164"
halavant.dll = "Digital Alpha XL 233/266 Family"
halxl.dll    = "Digital Alpha XL 300/366 Family"
hallx3.dll   = "Digital AlphaStation 255"
halflex.dll  = "DeskStation Technology UniFlex and Raptor 3 Systems"
halrawmp.dll = "Digital AlphaServer 4x00 5/xxx Family"
halrawmp.dll = "Digital AlphaServer 4100-5 - Multiprocessor"
halalp.dll   = "Aspen Alpine/Telluride"
haltimbr.dll = "Aspen Timberline/Summit"
halpinna.dll = "Digital AlphaStation 600A 5/500"
halpinna.dll = "Digital AlphaServer 1000 5/xxx Family"
halpinna.dll = "Digital AlphaServer 1000a 5/xxx Family"
hallego.dll  = "Digital Alpha 21064A PICMG SBC"
*halmiata.dll= "Digital Personal Workstation 433a, 500a"
```

Abbildung 2.2: HAL-Varianten

Aufgabe 2.1 *Informieren Sie sich in*
`http://www.dewassoc.com/support/win2000/tshoot_hal.htm`
über das Vorgehen, eine Windows-Installation für eine andere HW-Plattform zu modifizieren.

Weiterführende Literatur: http://www.operating-system.org/betriebssystem/_german/index.htm

2.2.2 Der Kern des Windows-Kernels

Der **KERN** ist der zentrale Bestandteil des Windows-Kernels. Er stellt die für alle weiteren Kernel-Module benötigte Funktionalität und Objekte bereit:

- Kontextwechsel von Prozessen
- Scheduling von Threads (Zeitscheibenwechsel, Interrupt-Behandlung, ...)
- Kontrollobjekte (z.B.: Prozessobjekte, deferred procedure calls — DCP —, asynchronous procedure calls — APC —, ...)
- DCP-Warteschlangen
- Dispatcher-Objekte wie Semaphore, Mutexe, Events, Stoppuhren, ...

2.2.3 Kernel-Module

Folgende **Dienste** werden von den Kernel-Modulen bereitgestellt:

- Prozess- und Auftragsverwaltung (Dispatcher; Resourceneigentum)
- Speicherverwaltung (einschließlich virtuellem Hauptspeicher)
- Threadverwaltung (Scheduler; CPU-Zeitscheiben)
- Geräteverwaltung
- Dateisysteme
- Sicherheitsmanagement (Authentifizierung)

Die Module (Manager) im Einzelnen:

Executive (=Kern-Oberteil)	**Erleuterung**
- E/A-Manager:	geräteunabhängige Ein- und Ausgabe, hier befinden sich die Gerätetreiber, z.B. für FAT, NTFS, ...
- Objektmanager:	verwaltet Prozesse, Threads, Dateien, Verzeichnisse, E/A-Geräte, Semaphoren, ... (virtuelle Adressblöcke im Kernel-Adressraum: Anlegen / Freigabe)
- Prozessmanager:	erzeugt/vernichtet Prozesse und Threads
- Speichermanager:	realisiert virtuellen Speicher, bildet virtuelle Seiten auf physikalische Seiten ab, realisiert Schutzregeln
- Sicherheitsmanager:	stellt "orange book C2 standard security"bereit; Regeln für Authentifizierung (Anmeldung am System), Zugriffskontrolle (z.B. müssen virtuelle Seiten anfangs mit Nullen (0) gefüllt werden, ...) , ...
- Cachemanager:	enthält Funktionen für die Bereitstellung von Datei-Puffern, ...
- PnP-Manager:	"Plug and Play"-Funktionalität für USB-, PCMCIA-Geräte, ...
- Powermanager:	realisiert "Energy Star"-Anforderungen (Standbymodus des Monitors, Akku-Überwachung, Notabschaltung des Notebooks, ...
- Konfigurationsmanager:	realisiert die Registry-Funktionalitäten: (Schlüssel, Value)-Paare hinzufügen, löschen, abfragen
- LPC-Manager: (Local Procedure Call)	stellt hocheffiziente Prozesskommunikationsfunktionen zwischen Prozessen und OS-Subsystemen zur Verfügung
- GDI: (graphical device interface)	stellt geräteunabhängige Funktionen für Monitor- und Druckfunktionalitäten bereit, realisiert den Fenster-Manager, enthält Treiber für Grafikkarten

Tabelle 2.1: Windows 2000 Kernel-Module

2.2.4 Systemdienste und Systemschnittstelle

Darüber liegt die Schicht der **Systemdienste**, die die Aufrufschnittstellen auf die Executive-Funktionalitäten auch den Subsystemen (Win32, OS/2, POSIX) bzw. den Benutzerprozessen bereitstellt (wrapper-Funktionen, WIN-API, Native API NTDLL.DLL):

http://www.sysinternals.com/ntw2k/info/ntdll.shtml

Aufgabe 2.2 *Erstellen Sie eine Liste der Funktionalitäten, die lediglich im native WIN-API, nicht jedoch im Win32-Subsystem verfügbar sind. Kommentieren Sie!*

2.2.5 Subsysteme

http://www.winprog.org/tutorial/

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/UCMGch02.asp

http://www.microsoft.com/windowsserversystem/sfu/productinfo/overview/default.mspx

http://winntposix.sourceforge.net/

Aufgabe 2.3 *Für welche Zwecke stellt Microsoft das POSIX-API zur Verfügung? Wo liegen im POSIX-API von Windows 2000 noch Defizite?*

2.3 System-Architektur von UNIX/Linux

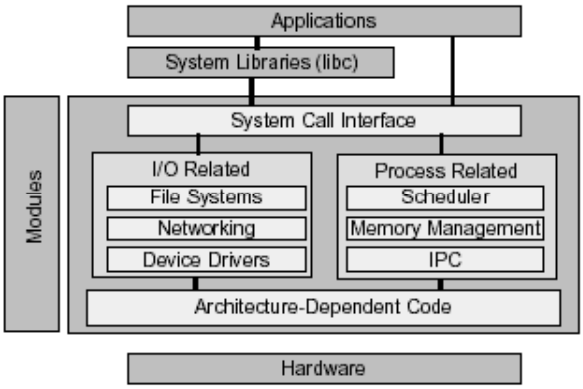


Abbildung 2.3: Linux-Architektur

Linux (und UNIX) besitzt einen monolitischen Kernel. Lediglich die Devicedriver werden als externe Module dynamisch nachgeladen.

Eine Einführung in den Linux-Kernel finden Sie unter:

http://cs-pub.bu.edu/fac/richwest/cs591_w1/notes/wk1.pdf
http://plg.uwaterloo.ca/~itbowman/CS746G/a1/
http://bitterberg.de/tilmann/kernel-intro-acrobat.pdf
http://cs.uml.edu/~cgould/

Die Darstellungen bei der Diskussion verschiedener Unix-Kernels weichen je nach Autoren-Intention etwas voneinander ab,

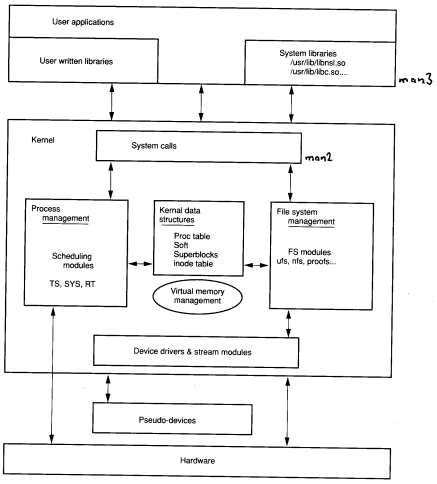


Abbildung 2.4: Solaris2-Architektur

(Vgl. auch: http://www.nacs.uci.edu/indiv/dwatanab/usenix/01/solaris_internals.pdf sowie http://www.solarisinternals.com/si/reading/t2-solaris-slides.pdf)

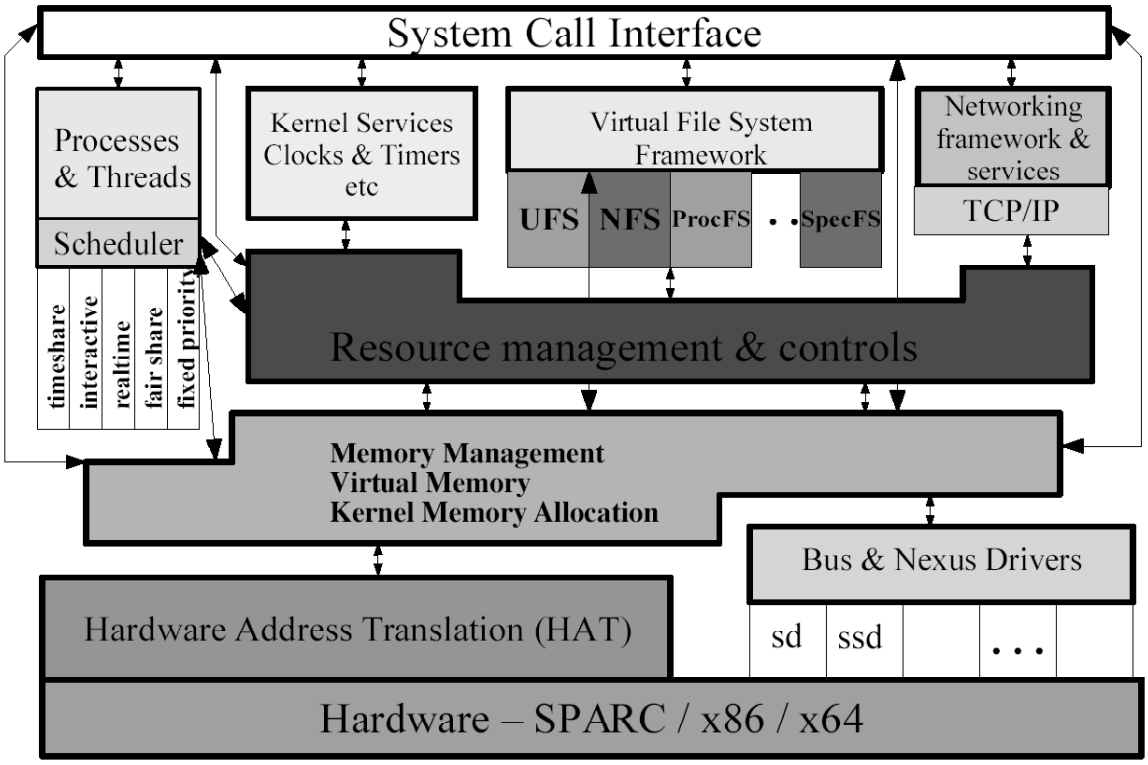


Abbildung 2.5: Solaris Kernel Overview

2.4 Virtuelle Betriebssysteme

Ein virtueller Computer (häufig mit Hilfe eines virtuellen Host-Betriebssystems realisiert) erlaubt Nutzern, auf einer einzigen Hardware mit einer Vielzahl von „virtuellen“ Computern mit unterschiedlichen Betriebssystemumgebungen, die sich alle gegeneinander lediglich über (virtuelle) Netzwerkinterfaces sehen können, zu arbeiten.
Repräsentative Beispiele sind:

- VMware: http://en.wikipedia.org/wiki/VMware
- VirtualPC: http://en.wikipedia.org/wiki/Virtual_PC
- Xen: http://en.wikipedia.org/wiki/Xen_%28virtual_machine_monitor%29
- Solaris 10 Zones:
  http://www.softpanorama.org/Solaris/zones.shtml und http://docs.sun.com/app/docs/doc/817-1592

Siehe auch: http://www.deatech.com/deatech/articles/linux_pc_vc.html
und http://en.wikipedia.org/wiki/Comparison_of_virtual_machines

Hardware-Unterstützung für virtuelle Maschinen nicht nur in Großrechnern:
http://www.heise.de/newsticker/result.xhtml?url=/newsticker/meldung/59885&words=Pacifica

2.5 Secure global desktop/Virtual network computing

Im virtual network computing trennt man den Arbeitsplatz (Tastatur/Maus/...) über ein Netzwerk vom Computer:

- UNIX X-Window (UNIX (native), Windows: XWin32, ...)
- Windows Terminal Services / RemoteDesktopProtocol (UNIX: Rdesktop/Citrix, Windows: native/Citrix, ...)
- VNC (http://www.realvnc.com/what.html, „Secure Global Desktop“, ...)
- Nachfolger von X-Terminals: Sun Ray 1

Siehe dazu auch: http://en.wikipedia.org/wiki/Thin_client

2.6 Linken und Laden: statische und shared Bibliotheken

Der Systemcall `execve()` bringt ein Executable zum Ablauf, indem er

a) die effektive User- und Gruppen-ID des laufenden Prozesses auf die User- und/oder Gruppen-ID des Executables ändert, falls das Executable-File das suid- und/oder sgid-Flag gesetzt hat,

b) das Executable in den Speicher läd und

b) den Loader die benötigten shared Bibliotheken laden und mit dem Executable verlinken läßt.

2.6.1 Linken

Programme können mittels

```
gcc -static hw.c -o hw
```

statisch oder mittels

```
gcc hw.c -o hw2
```

dynamisch mit den benötigten Runtime-Bibliotheksfunktionen verbunden (gelinkt) werden. Im ersten Falle werden alle benötigten Bibliotheksfunktionen direkt mit dem Code der Datei `hw.c` zusammen im Executable `hw` abgelegt. Beim Linken werden also schon alle externen Bezüge aufgelöst. Die Datei `hw` ist deshalb auch gecht groß:

```
buhl@wminf11:~/prg/OS> ls -al hw
-rwxr-xr-x  1 buhl users 2215534 2005-05-24 16:03 hw
buhl@wminf11:~/prg/OS> size hw
   text    data     bss     dec     hex filename
 373281    3188    4416  380885   5cfd5 hw
```

Dynamisch gelinkte Executables enthalten lediglich symbolische Referenzen auf die benötigten Bibliotheksfunktionen, die beim Laden (mittels `execve()`) aufgelöst werden, weshalb dynamisch gelinkte Executables deutlich weniger Plattenplatz benötigen:

```
buhl@wminf11:~/prg/OS> ls -al hw2
-rwxr-xr-x  1 buhl users 8996 2005-05-24 15:37 hw2
buhl@wminf11:~/prg/OS> size hw2
   text    data     bss     dec     hex filename
   1002     256       4    1262     4ee hw2
buhl@wminf11:~/prg/OS> ldd hw2
        linux-gate.so.1 =>  (0xffffe000)
        libc.so.6 => /lib/tls/libc.so.6 (0x4003a000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

`ld-linux.so.2` ist der zur Ladezeit nötige Linker-Anteil, d.h. der Loader. Die zum Auflösen durch den Loader nötigen Informationen stehen in der Symboltabelle der Datei `hw2`:

```
buhl@wminf11:~/prg/OS> nm hw2
0804962c A __bss_start
08048324 t call_gmon_start
0804962c b completed.1
08049530 d __CTOR_END__
0804952c d __CTOR_LIST__
08049620 D __data_start
```

```
08049620 W data_start
080484c0 t __do_global_ctors_aux
08048350 t __do_global_dtors_aux
08049624 D __dso_handle
08049538 d __DTOR_END__
08049534 d __DTOR_LIST__
08049540 D _DYNAMIC
0804962c A _edata
08049630 A _end
080484e4 T _fini
0804952c A __fini_array_end
0804952c A __fini_array_start
08048500 R _fp_hw
08048390 t frame_dummy
08048528 r __FRAME_END__
0804960c D _GLOBAL_OFFSET_TABLE_
         w __gmon_start__
080484b8 T __i686.get_pc_thunk.bx
080482b0 T _init
0804952c A __init_array_end
0804952c A __init_array_start
08048504 R _IO_stdin_used
0804953c d __JCR_END__
0804953c d __JCR_LIST__
         w _Jv_RegisterClasses
080483f0 T __libc_csu_fini
08048460 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080483bc T main
08049628 d p.0
         U printf@@GLIBC_2.0
08048300 T _start
```

Weitere interessante Tools sind `objdump` und `readelf`:

```
buhl@wminf11:~/prg/OS> objdump -S hw2

hw2:     file format elf32-i386

Disassembly of section .init:

080482b0 <_init>:
 80482b0:       55                      push   %ebp
 80482b1:       89 e5                   mov    %esp,%ebp
 80482b3:       83 ec 08                sub    $0x8,%esp
 80482b6:       e8 69 00 00 00          call   8048324 <call_gmon_start>
 80482bb:       e8 d0 00 00 00          call   8048390 <frame_dummy>
 80482c0:       e8 fb 01 00 00          call   80484c0 <__do_global_ctors_aux>
 80482c5:       c9                      leave
 80482c6:       c3                      ret
Disassembly of section .plt:
```

```
080482c8 <__libc_start_main@plt-0x10>:
 80482c8:       ff 35 10 96 04 08       pushl  0x8049610
 80482ce:       ff 25 14 96 04 08       jmp    *0x8049614
 80482d4:       00 00                   add    %al,(%eax)
        ...

080483bc <main>:
 80483bc:       55                      push   %ebp
 80483bd:       89 e5                   mov    %esp,%ebp
 80483bf:       83 ec 08                sub    $0x8,%esp
 80483c2:       83 e4 f0                and    $0xfffffff0,%esp
 80483c5:       b8 00 00 00 00          mov    $0x0,%eax
 80483ca:       83 c0 0f                add    $0xf,%eax
 80483cd:       83 c0 0f                add    $0xf,%eax
 80483d0:       c1 e8 04                shr    $0x4,%eax
 80483d3:       c1 e0 04                shl    $0x4,%eax
 80483d6:       29 c4                   sub    %eax,%esp
 80483d8:       83 ec 0c                sub    $0xc,%esp
 80483db:       68 08 85 04 08          push   $0x8048508
 80483e0:       e8 03 ff ff ff          call   80482e8 <printf@plt>
 80483e5:       83 c4 10                add    $0x10,%esp
 80483e8:       b8 00 00 00 00          mov    $0x0,%eax
 80483ed:       c9                      leave
 80483ee:       c3                      ret
 80483ef:       90                      nop
        ...
Disassembly of section .fini:

080484e4 <_fini>:
 80484e4:       55                      push   %ebp
 80484e5:       89 e5                   mov    %esp,%ebp
 80484e7:       53                      push   %ebx
 80484e8:       e8 00 00 00 00          call   80484ed <_fini+0x9>
 80484ed:       5b                      pop    %ebx
 80484ee:       81 c3 1f 11 00 00       add    $0x111f,%ebx
 80484f4:       50                      push   %eax
 80484f5:       e8 56 fe ff ff          call   8048350 <__do_global_dtors_aux>
 80484fa:       59                      pop    %ecx
 80484fb:       5b                      pop    %ebx
 80484fc:       c9                      leave
 80484fd:       c3                      ret
```

Der Aufbau der ELF-Datei `hw2`:

```
buhl@wminf11:~/prg/OS> objdump -h hw2

hw2:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .interp       00000013  08048134  08048134  00000134  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  08048148  08048148  00000148  2**2
```

```
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 2 .note.SuSE     00000018  08048168  08048168  00000168  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 3 .hash          0000002c  08048180  08048180  00000180  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 4 .dynsym        00000060  080481ac  080481ac  000001ac  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 5 .dynstr        00000060  0804820c  0804820c  0000020c  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 6 .gnu.version   0000000c  0804826c  0804826c  0000026c  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 7 .gnu.version_r 00000020  08048278  08048278  00000278  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 8 .rel.dyn       00000008  08048298  08048298  00000298  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 9 .rel.plt       00000010  080482a0  080482a0  000002a0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
10 .init          00000017  080482b0  080482b0  000002b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
11 .plt           00000030  080482c8  080482c8  000002c8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
12 .text          000001e4  08048300  08048300  00000300  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
13 .fini          0000001a  080484e4  080484e4  000004e4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
14 .rodata        00000026  08048500  08048500  00000500  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
15 .eh_frame      00000004  08048528  08048528  00000528  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
16 .ctors         00000008  0804952c  0804952c  0000052c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
17 .dtors         00000008  08049534  08049534  00000534  2**2
                  CONTENTS, ALLOC, LOAD, DATA
18 .jcr           00000004  0804953c  0804953c  0000053c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
19 .dynamic       000000c8  08049540  08049540  00000540  2**2
                  CONTENTS, ALLOC, LOAD, DATA
20 .got           00000004  08049608  08049608  00000608  2**2
                  CONTENTS, ALLOC, LOAD, DATA
21 .got.plt       00000014  0804960c  0804960c  0000060c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
22 .data          0000000c  08049620  08049620  00000620  2**2
                  CONTENTS, ALLOC, LOAD, DATA
23 .bss           00000004  0804962c  0804962c  0000062c  2**2
                  ALLOC
24 .comment       00000111  00000000  00000000  0000062c  2**0
                  CONTENTS, READONLY
25 .debug_aranges 00000098  00000000  00000000  00000740  2**3
                  CONTENTS, READONLY, DEBUGGING
26 .debug_pubnames 0000005f  00000000  00000000  000007d8  2**0
                  CONTENTS, READONLY, DEBUGGING
27 .debug_info    00000304  00000000  00000000  00000837  2**0
```

```
                  CONTENTS, READONLY, DEBUGGING
 28 .debug_abbrev 0000011d  00000000  00000000  00000b3b  2**0
                  CONTENTS, READONLY, DEBUGGING
 29 .debug_line   0000023f  00000000  00000000  00000c58  2**0
                  CONTENTS, READONLY, DEBUGGING
 30 .debug_frame  00000058  00000000  00000000  00000e98  2**2
                  CONTENTS, READONLY, DEBUGGING
 31 .debug_str    00000125  00000000  00000000  00000ef0  2**0
                  CONTENTS, READONLY, DEBUGGING
 32 .debug_loc    00000047  00000000  00000000  00001015  2**0
                  CONTENTS, READONLY, DEBUGGING
 33 .debug_ranges 00000018  00000000  00000000  0000105c  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Die Relokationsinformationen aus `hw2`:

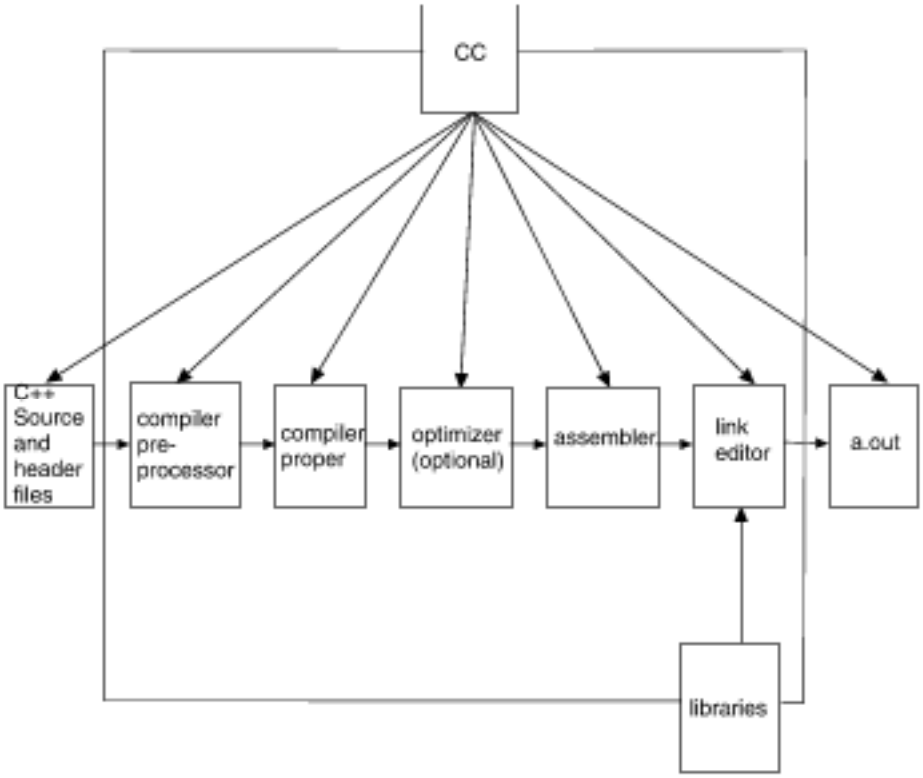
```
buhl@wminf11:~/prg/OS> readelf -d hw2

Dynamic section at offset 0x540 contains 20 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000c (INIT)                       0x80482b0
 0x0000000d (FINI)                       0x80484e4
 0x00000004 (HASH)                       0x8048180
 0x00000005 (STRTAB)                     0x804820c
 0x00000006 (SYMTAB)                     0x80481ac
 0x0000000a (STRSZ)                      96 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x804960c
 0x00000002 (PLTRELSZ)                   16 (bytes)
 0x00000014 (PLTREL)                     REL
 0x00000017 (JMPREL)                     0x80482a0
 0x00000011 (REL)                        0x8048298
 0x00000012 (RELSZ)                      8 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffe (VERNEED)                    0x8048278
 0x6fffffff (VERNEEDNUM)                 1
 0x6ffffff0 (VERSYM)                     0x804826c
 0x00000000 (NULL)                       0x0
```

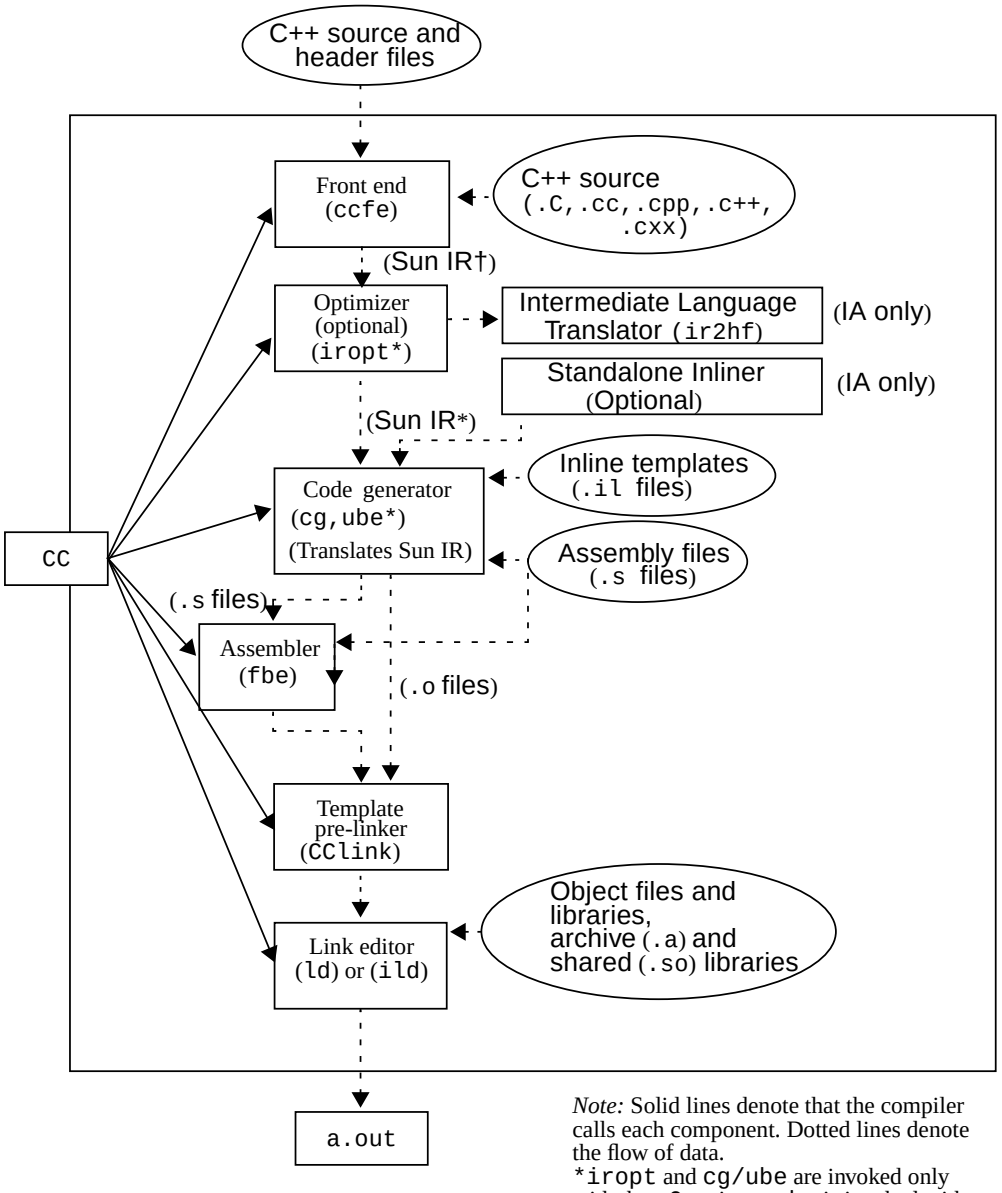
Literatur

- http://www.lurklurk.org/linkers/linkers.html
- http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
- http://andercheran.aiind.upv.es/toni/prog/ELF_From_The_Programmers_Perspective.ps.gz
- http://docs.sun.com/app/docs/doc/816-1386

2.6.2 Compilationsphasen

Der C- und der Java-Compilationsprozess:
http://www.acm.uiuc.edu/sigmil/RevEng/ch02.html





Note: Solid lines denote that the compiler calls each component. Dotted lines denote the flow of data.
`*iropt` and `cg/ube` are invoked only with the `-O` option. `yabe` is invoked with `-g`.

†Sun IR = Sun's Intermediate Representation

FIGURE 2-1 The Compilation Process

2.6.3 Statisches und dynamisches Linken im Vergleich

Vorteile/Nachteile des statische Linkens von Applikationen:

- hoher Plattenplatzbedarf jedes statisch gelinkten Binaries
- hoher Hauptspeicherbedart jedes statisch gelinkten Binaries
- Mehrfachresidenz von (Bibliotheks-)Funktionscode im Hauptspeicher
- zur Laufzeit kein Nachladen-/-linken nötig
- Unabhängigkeit des statisch gelinkten Binaries von der Version der im System vorhandenen Bibliotheken
- Bei verbesserten (gepatchten) Bibliotheken ist ein erneutes Linken der statischen Binaries nötig, damit diese Verbesserungen wirksam werden.

2.6.4 Generierung einer statischen Bibliothek libmylib1.a

Mit gnucc

```
gcc -Wall -g -c libmylib1.c
ar rcs libmylib1.a libmylib1.o
...
gcc -g -static demo_using_libmylib1.c -L. -lmylib1
```

bzw. mit Sun CC:

```
CC +w2 -g -c libmylib1.cc
CC -xar -o libmylib1.a libmylib1.o
...
CC  -g -Bstatic demo_using_libmylib1.cc -L. -lmylib1
```

2.6.5 Generierung einer shared Bibliothek libmylib1.so.0.0

Mit gnucc

```
gcc -fPIC -Wall -g -c libmylib1.c
gcc -g -shared -Wl,-soname,libmylib1.so.0 -o libmylib1.so.0.0 libmylib1.o -lc
ln -sf libmylib1.so.0.0 libmylib1.so.0
ln -sf libmylib1.so.0 libmylib1.so
export LD_LIBRARY_PATH=.:${LD_LIBRARY_PATH}
...
gcc -g demo_using_libmylib1.c -L. -lmylib1
```

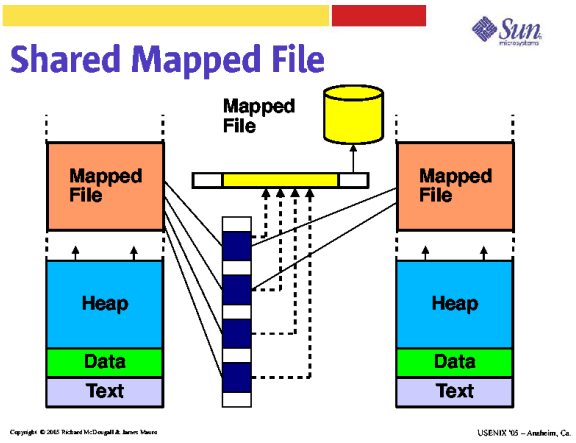
bzw. mit Sun cCC:

```
CC -KPIC +w2 -g -c libmylib1.cc
CC -KPIC -G -h libmylib1.so.0 -o libmylib1.so.0.0 libmylib1.o
ln -sf libmylib1.so.0.0 libmylib1.so.0
ln -sf libmylib1.so.0 libmylib1.so
export LD_LIBRARY_PATH=.:${LD_LIBRARY_PATH}
...
CC -g demo_using_libmylib1.c -L. -lmylib1
```

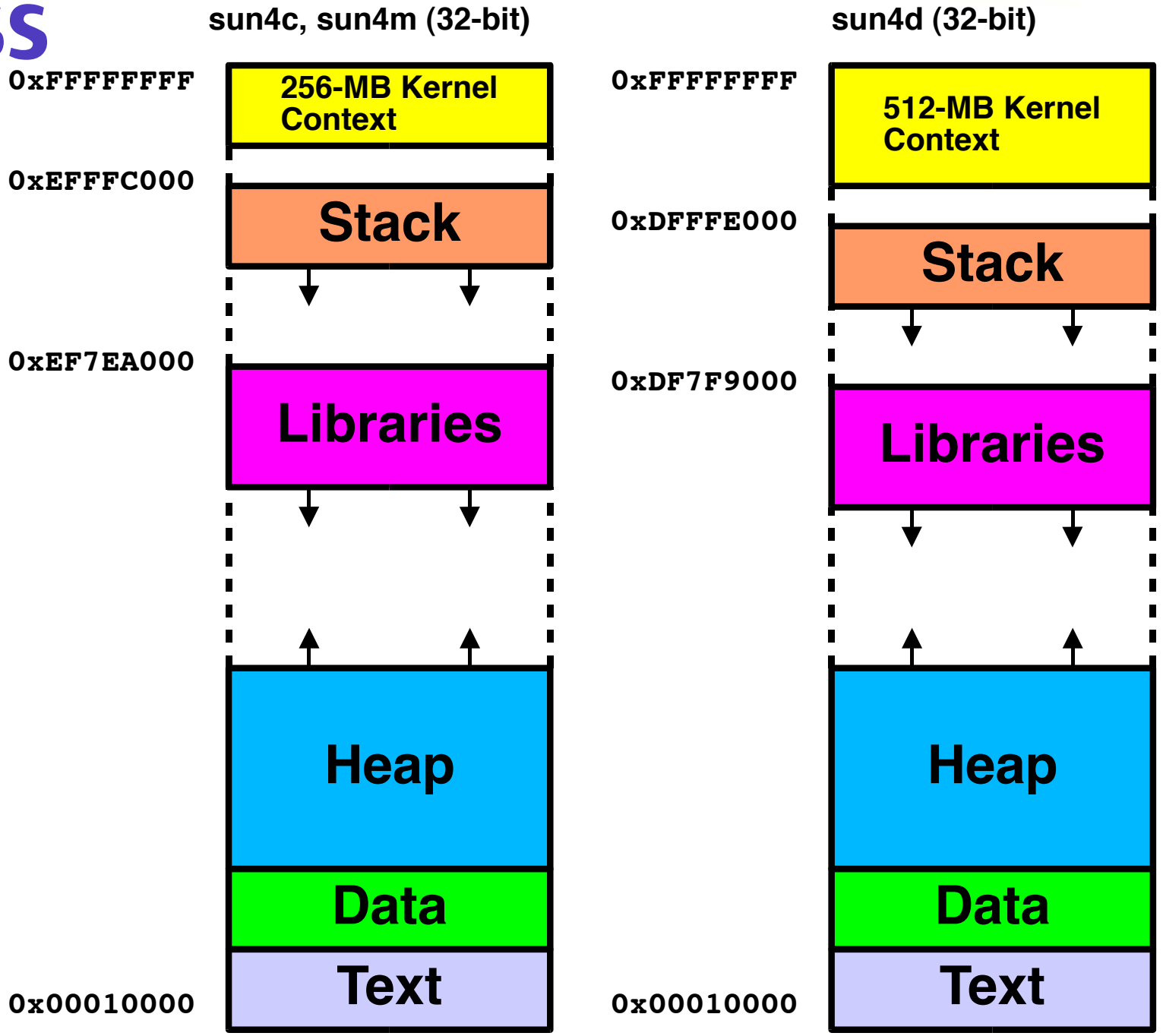
2.6.6 Memory-Layout von Prozessen

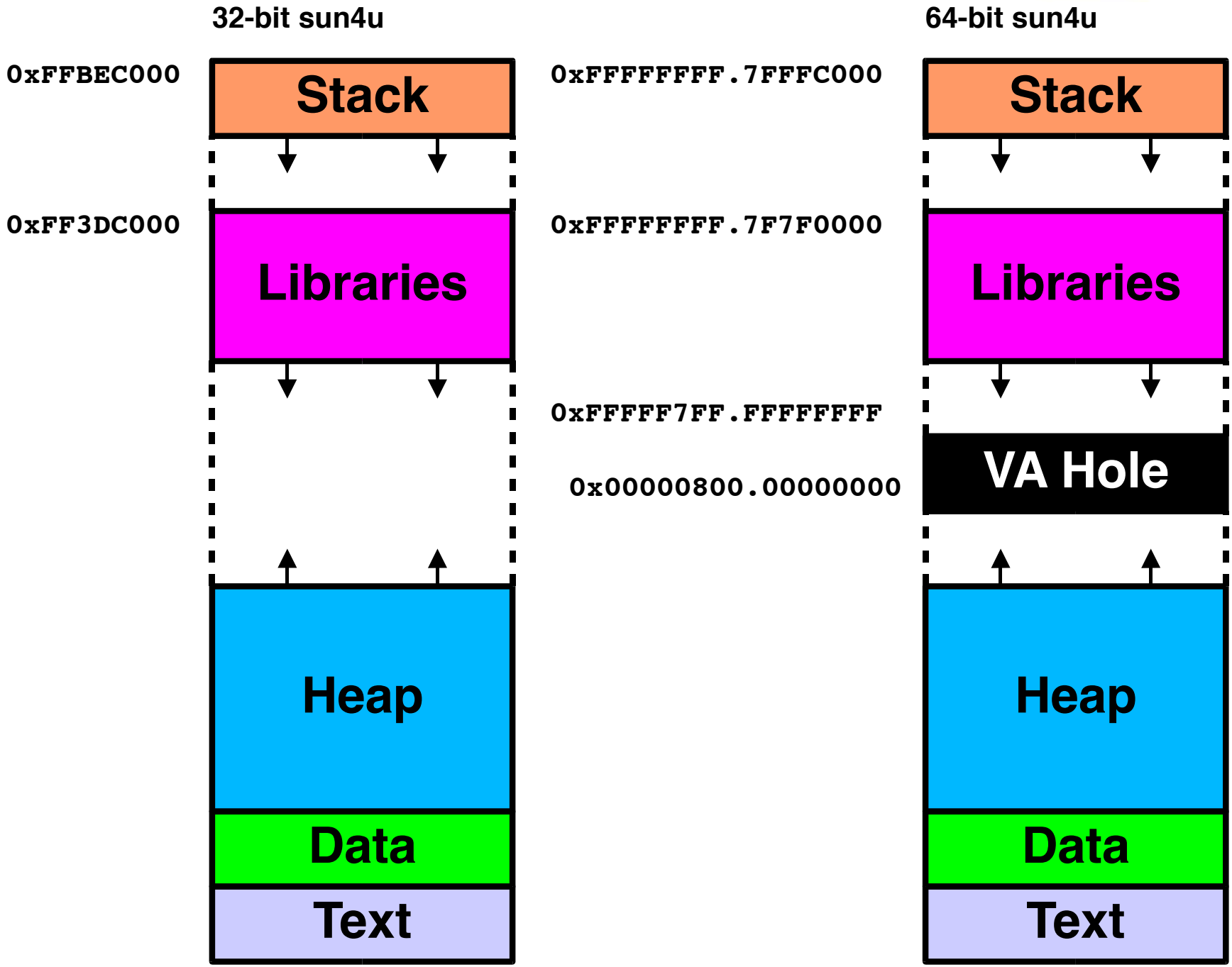
Unter Windows:
http://support.microsoft.com/kb/q125691/
http://i30www.ira.uka.de/teaching/coursedocuments/61/Mirko-Taenzler_Windows-Address-Space.pdf
http://www.altusnet.com/lti/160web/addressSpace.htm

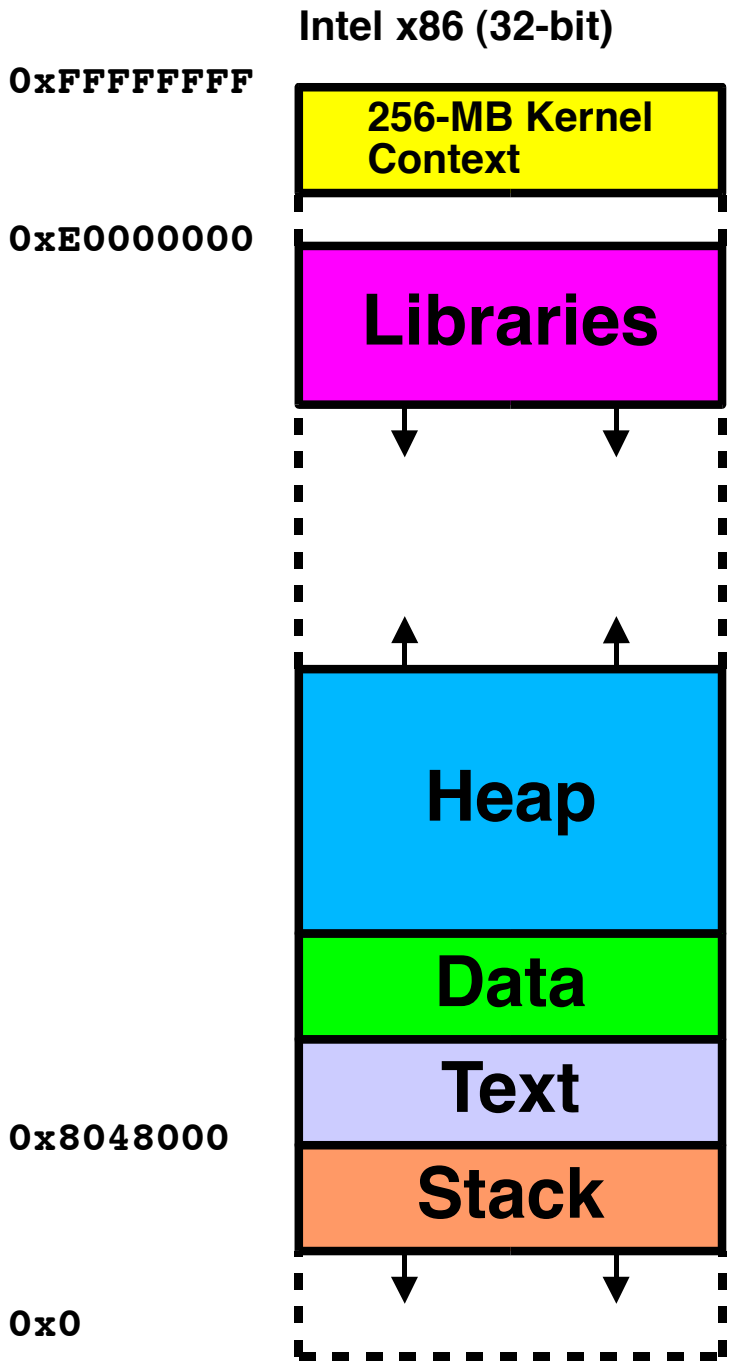
Unter UNIX:
http://docs.sun.com/app/docs/doc/806-0477/6j9r2e2b9?q=stack+bias&a=view
http://www.nacs.uci.edu/indiv/dwatanab/usenix/01/solaris_internals.pdf
http://www.solarisinternals.com/si/reading/t2-solaris-slides.pdf



Address Space

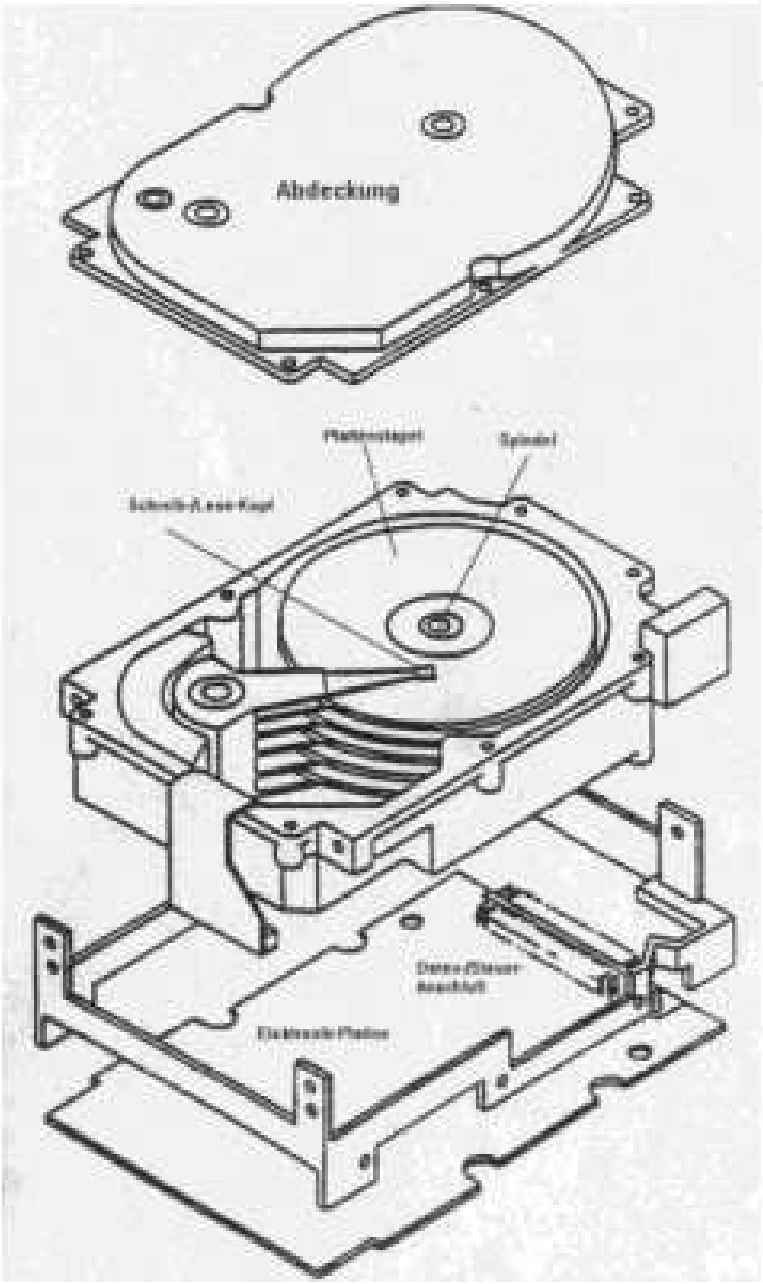






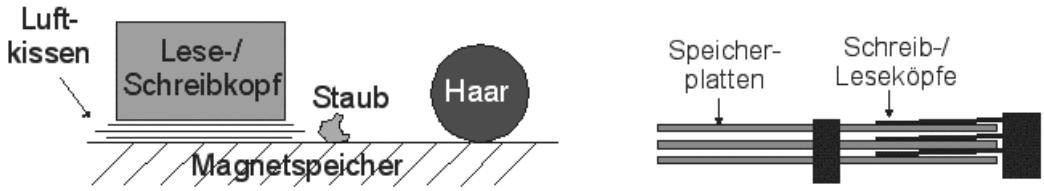
2.7 Lowlevel-Aufbau von Festplatten

2.7.1 Physikalischer Aufbau

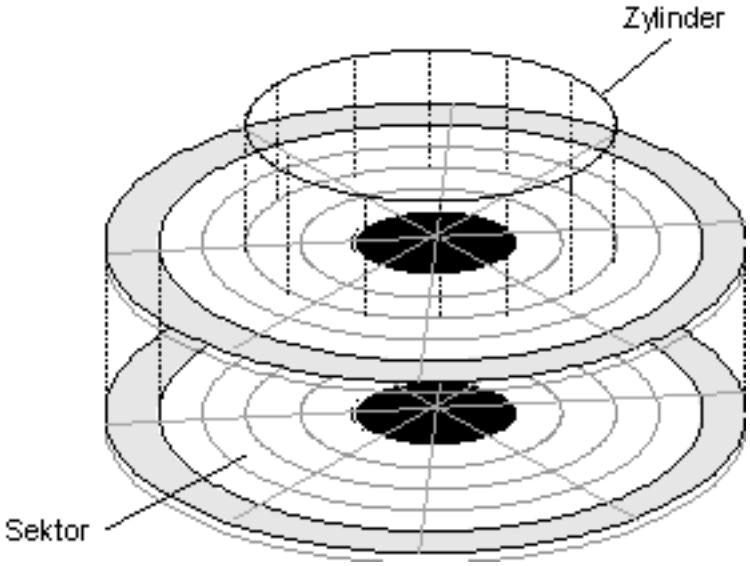


Schematiche Darstellung einer Festplatte

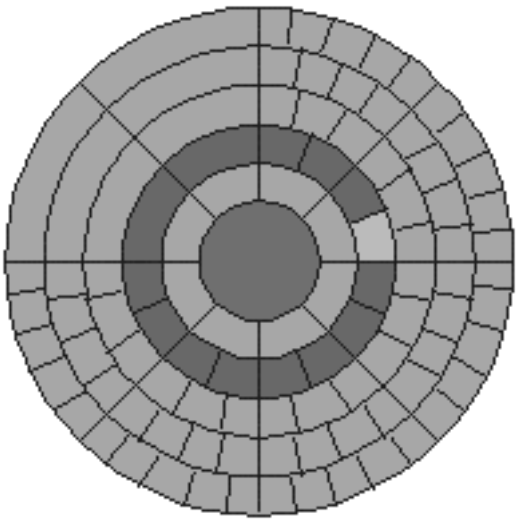
Die Schreib-/Leseköpfe setzen im Gegensatz zu einer DISKETTE **nicht** auf, sondern *schweben* über der Festplatte.



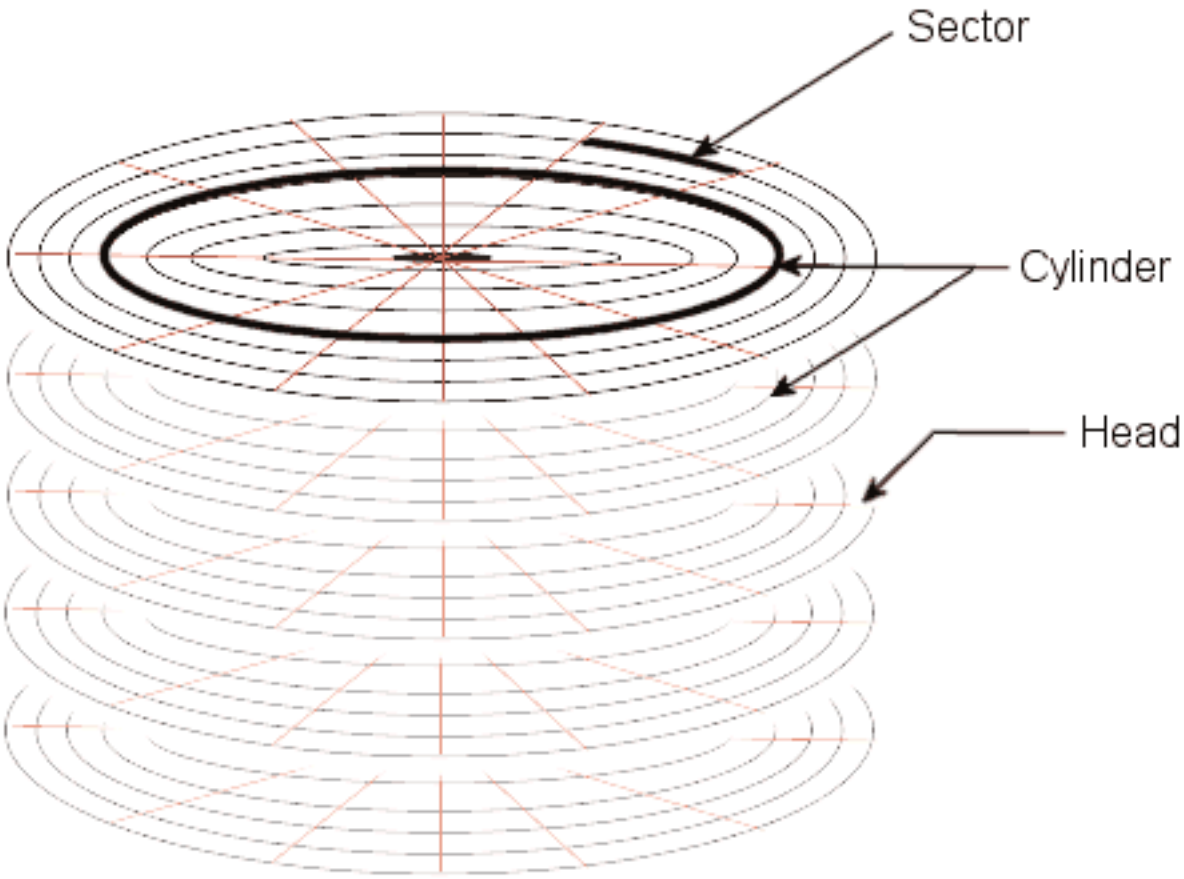
Konzentrische Kreise, die einzelen Spuren (TRACKS) werden mit Informationen beschrieben.



Jede Spur ist in eine Anzahl von Sektoren unterteilt:



Gleichzeitig sind immer alle gleichweit von der Drehachse entfernten Spuren des Plattenstapels verfügbar (Zylinder).



Eine `lowlevel`-Formatierung nimmt die Unterteilung der Festplatte (softsectored) vor:

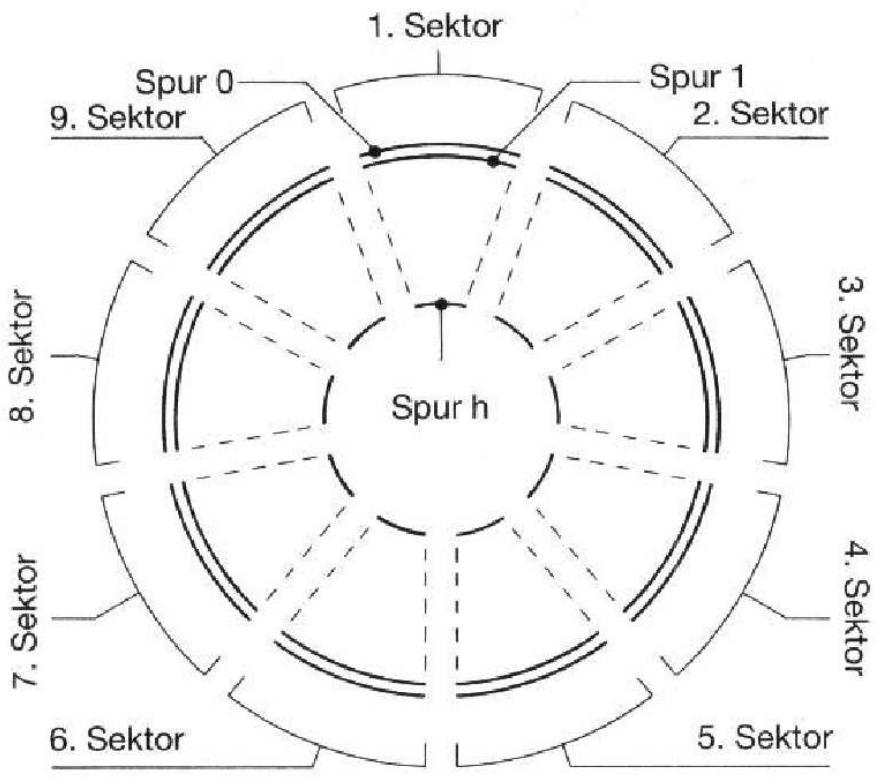


Abbildung 2.6: Spuren und Sektoren einer Festplatte

2.7.2 Die magnetische Unterteilung der Festplatte

Die low-level-Formatierung schreibt folgenden Magnetmarken-Verwaltungsrahmen auf die Festplatte:

Bytes	Name	Beschreibung
16	POST INDEX GAP	Alle 4Eh; am Spurbeginn nach der Indexmarkierung
Die folgenden Sektor-Daten (die in dieser Tabelle gezeigt werden) werden für eine MFM-kodierte Spur 17 mal wiederholt.		
13	ID VFO LOCK	Alle 00h; synchronisieren den VFO für die Sektor-ID-Erkennung
1	SYNC BYTE	A1h; benachrichtigt den Controller darüber , dass die Daten beginnen
1	ADDRESS MARK	FEh; Beginn-Marke der Sektor-Addressdaten
2	CYLINDER NUMBER	Zylindernummer
1	HEAD NUMBER	Kopfnummer
1	SECTOR NUMBER	Sektornummer
2	CRC	CRC-Prüfsumme und End-Marke der Sektor-Addressdaten
3	WRITE TURN-ON GAP	00h; geschrieben, um die Address-Daten von den Daten selbst sauber zu trennen
13	DATA SYNC VFO LOCK	Alle 00h; synchronisiert den VFO für die Daten des Sektors
1	SYNC BYTE	A1h; benachrichtigt den Controller darüber, dass Daten beginnen
1	ADDRESS MARK	F8h; definiert, Beginn-Marke der Sektor-Daten
512	DATA	die Sektordaten selbst
2	CRC	CRC-Prüfsumme der Sektordaten
3	WRITE TURN-OFF GAP	00h; geschrieben, um Daten sauber zu trennen
15	INTER-RECORD GAP	Alle 00h; Puffer, um Spindel-Geschwindigkeitsschwankungen ausgleichen zu können
693	PRE-INDEX GAP	Alle 4Eh; am Spur-Ende vor der Index-Marke

3 Partitionen und Dateisysteme

3.1 Virtuelle Festplatten

3.1.1 Zweck von Festplatten-Partitionierung

Für einige Zwecke sollte man eine Festplatte in mehrere logische Festplatten unterteilen; sie sieht dann dem Betriebssystem gegenüber wie mehrere (physikalich vorhandene) Festplatten aus:

- Vorspiegelung mehrerer kleiner Festplaten, wenn die Gesamtkapazität sonst für das Betriebsystem/Dateisystem unbrauchbar groß ist (FAT16-Dateisysteme konnten z.B. nur max. 2 GB groß sein).
- Gleichzeitige Installierung mehrerer Betriebsysteme auf einer Festplatte.
- Logische Trennung der Bereiche einer Festplatte nach ihrer Funktion: zum Beispiel Partitionen für das Betriebssystem, den Swapping-Bereich, die Anwenderprogramme, die Datenbereiche der einzelnen Benutzer, ...
- Weitergehende logische Trennung der Bereiche des Betrienssystems: Loggingdatenbereiche, Bereiche für optionale Software, Temporärbereiche (etwa 800 MB für das Image einer CD bzw. 5 GB bzw. 8.5 GB für ein DVD-Image),...
- Anlegen einer gemeinsamen Datenpartition (z.B. FAT) zum Datenaustausch zwischen verschiedenen installierten Betrienssystemen.
- Anlegen einer dedizierten nur zum Swapping benutzten Partition.
- Einrichten einer temporären Scratch-Partition.
- Einrichten eine Backup-Partition (Backup durch manuelles Spiegeln).
- Partitionen für Software-RAID-Volumes (Spiegeln, Stripes)
- ...

Man nennt solche logischen Festplatten **Partitionen**. Eine Festplatte wird für die genannten Zwecke *partitioniert*.

3.1.2 Partitionieren unter UNIX/SOLARIS

SOLARIS-Festplatten können in sieben Partitionen unterteilt werden, von denen eine (Nummer 2) in der Regel als Pseudopartition die ganze Festplatte erfaßt.

Unter SOLARIS partitioniert man eine Platte mit dem Befehl `format`:

```
% format>
Serching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c0t0d0 <SUN4.2G cyl 3880 alt 2 hd 16 sec 135>
          /pci@1f,4000/scsi@3/sd@0,0
       1. c0t8d0 <SEAGATE-ST336607LC-0003 cyl 49780 alt 2 hd 2 sec 720>
          /pci@1f,4000/scsi@3/sd@8,0
       2. c0t9d0 <SUN18G cyl 7506 alt 2 hd 19 sec 248>
          /pci@1f,4000/scsi@3/sd@9,0
       3. c0t10d0 <SEAGATE-ST318404LC-HP02 cyl 14078 alt 2 hd 6 sec 421>
          /pci@1f,4000/scsi@3/sd@a,0
       4. c0t11d0 <SEAGATE-ST3146807LC-0004 cyl 49780 alt 2 hd 8 sec 720>
          /pci@1f,4000/scsi@3/sd@b,0
       5. c0t12d0 <SEAGATE-ST3146807LC-0004 cyl 49780 alt 2 hd 8 sec 720>
          /pci@1f,4000/scsi@3/sd@c,0
Specify disk (enter its number): 5
selecting c0t12d0
[disk formatted]


FORMAT MENU:
FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        repair     - repair a defective sector
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        verify     - read and display labels
        save       - save new disk/partition definitions
        inquiry    - show vendor, product and revision
        volname    - set 8-character volume name
        !<cmd>     - execute <cmd>, then return
        quit
```

Abbildung 3.1: Partitionieren einer Solaris-Festplatte mittels **format**

```
PARTITION MENU:
        0      - change `0' partition
        1      - change `1' partition
        2      - change `2' partition
        3      - change `3' partition
        4      - change `4' partition
        5      - change `5' partition
        6      - change `6' partition
        7      - change `7' partition
        select - select a predefined table
        modify - modify a predefined partition table
        name   - name the current table
        print  - display the current table
        label  - write partition map and label to the disk
        !<cmd> - execute <cmd>, then return
        quit
partition> pr
Current partition table (original):
Total disk cylinders available: 49780 + 2 (reserved cylinders)

Part      Tag    Flag     Cylinders         Size            Blocks


  0 unassigned    wm       0               0          (0/0/0)           0
  1 unassigned    wu       0               0          (0/0/0)           0
  2     backup    wu       0 - 49779       136.72GB   (49780/0/0) 286732800
  3 unassigned    wm       0               0          (0/0/0)           0
  4 unassigned    wm       0               0          (0/0/0)           0
  5 unassigned    wm       0               0          (0/0/0)           0
  6 unassigned    wm       0               0          (0/0/0)           0
  7 unassigned    wm       0 - 49779       136.72GB   (49780/0/0) 286732800

partition> q
format> inq
Vendor:   SEAGATE
Product:  ST3146807LC
Revision: 0004
```

Abbildung 3.2: Partitionieren einer Solaris-Festplatte mittels **format** (Fortsetzung)

3.1.3 Windows-Partitionen

Windows-Festplatten erlaubten zunächst maximal vier Partitionen. Später erweiterte man das zu einem Schema von drei primären und einer erweiterten Partition, wobei die erweiterte Partition viele Unterpartitionen besitzen darf.

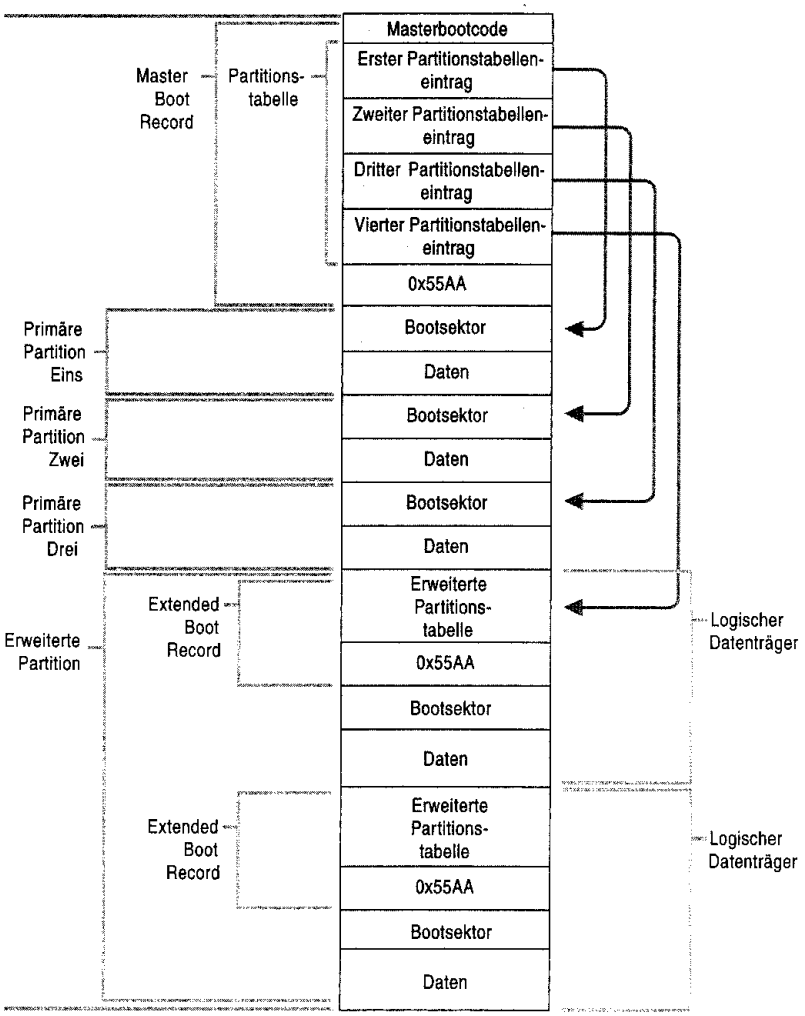
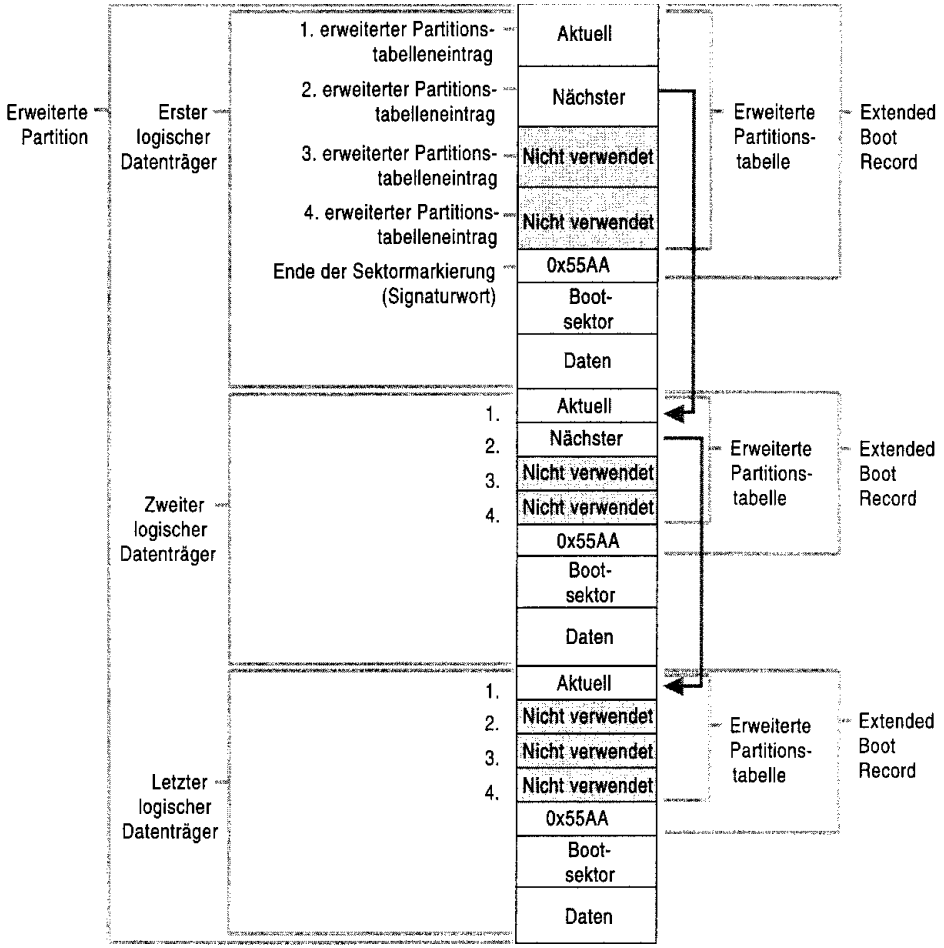


Abbildung 3.3: Vier primäre Partitionen


Erweiterte Partition
Erster logischer Datenträger
1. erweiterter Partitions-tabelleneintrag
2. erweiterter Partitions-tabelleneintrag
3. erweiterter Partitions-tabelleneintrag
4. erweiterter Partitions-tabelleneintrag
Ende der Sektormarkierung (Signaturwort)
Aktuell
Nächster
Nicht verwendet
Nicht verwendet
0x55AA
Boot-sektor
Daten
Erweiterte Partitions-tabelle
Extended Boot Record
Zweiter logischer Datenträger
1.
2.
3.
4.
Aktuell
Nächster
Nicht verwendet
Nicht verwendet
0x55AA
Boot-sektor
Daten
Erweiterte Partitions-tabelle
Extended Boot Record
Letzter logischer Datenträger
1.
2.
3.
4.
Aktuell
Nicht verwendet
Nicht verwendet
Nicht verwendet
0x55AA
Boot-sektor
Daten
Erweiterte Partitions-tabelle
Extended Boot Record

Partitionen werden durch eine *maggische Zahl* (die SYSTEM-ID) gekennzeichnet, die den Typ des auf der Partition installierten Dateisystems kennzeichnet:

EINTRAG	ERKLÄHRUNG
0x01	Primäre FAT12-Partition oder logisches Laufwerk (weniger als 32.680 Sektoren auf dem Datenträger)
0x04	FAT16-Partition oder logisches Laufwerk (32.680 - 65.535 Sektoren oder 16 MB - 33 MB
0x05	Erweiterte Partition
0x06	BIGDOS FAT16-Partition oder logisches Laufwerk (33 MB - 4 GB)
0x07	Installierbares Dateisystem (NTFS-Partition oder logisches Laufwerk)
0x0B	FAT32-Partition oder logisches Laufwerk
0x0C	FAT32-Partition oder logisches Laufwerk mit Verwendung von BIOS INT 13h-Erweiterung
0x0E	BIGDOS FAT16-Partition oder logisches Laufwerk mit Verwendung von BIOS INT 13h-Erweiterungen
0x0F	Erweiterte Partition mit Verwendung von BIOS INT 13h-Erweiterungen
0x12	EISA-Partition
0x42	Datenträger einer dynamischer Festplatte
0x86	Fehlertolerante veraltete FAT16-Festplatte*
0x87	Fehlertolerante veraltete NTFS-Festplatte*
0x8B	Fehlertoleranter veralteter Datenträger mit FAT32-Formatierung*
0x8C	Fehlertoleranter veralteter Datenträger mit Verwendung von BIOS INT 13h-Erweiterung und FAT32-Formatierung*

Tabelle 3.1: Datenträgertypen

Mit einem Sternchen (*) gekennzeichnete Partitionstypen können auch für nicht fehlertolerante Konfigurationen wie Stripset-Datenträger und übergreifende Datenträger verwendet werden.

Eine aktuelle neuere Übersicht der genutzten SYSTEM-IDs finden Sie unter:

http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

Bytewerte für Offset	Feld-länge	Beispiel-wert	Feldnahme und Definition
0x01BE	BYTE	0x80	**Boot Indikator**: Kennzeichnet ob es sich bei dem Datenträger um eine aktive Partition handelt. Gültige Werte sind: • 00. Keine Verwendung für den Startvorgang • 80. Aktive Partition
0x01BF	BYTE	0x01	**Sarting Head**
0x01C0	6 Bits	0x01*	**Starting Sector**: Es werden nur die Bits 0...5 verwendet. Die oberen zwei Bits, 6 und 7, werden durch das Feld Startzylinder mitverwendet.
0x01C1	10 Bits	0x00*	**Starting Cylinder**: Verwendet zusätzlich ein Byte zu den oberen 2 Bits des Startsektorfeldes zur Bildung des Zylinderwertes. Der Startzylinder ist eine 10-Bit-Nummer mit einem maximalen Wert von 1023.
0x01C2	BYTE	0x07	**System ID**: Definiert den Datenträgertyp. Siehe hierzu Tabelle **Datenträgertypen**.
0x01C3	BYTE	0xFE	**Ending Head**
0x01C4	6 Bits	0xBF*	**Ending sector**: Es werden nur die Bits 0...5 verwendet. Die oberen zwei Bits, 6 und 7, werden durch das Feld Endzylinder mitverwendet.
0x01C5	10 Bits	0x09*	**Ending Cylinder**: Verwendet zusätzlich ein Byte zu den oberen 2 Bits des Endsektorfeldes zur Bildung des Zylinderwertes. Der Endzylinder ist eine 10-Bit-Nummer mit einem maximalen Wert von 1023.
0x01C6	DWORD	0x3F000000	**Relative Sectors**: Das Offset vom Beginn der Festplatte bis zum Beginn der Partition gezählt in Sektoren.
0x01CA	DWORD	0x4BF57F00	**Total Sectors**: Die Gesamtzahl der Sektoren der Partition.

Tabelle 3.2: Partitionstabellenfelder

Ein BYTE umfasst 8 Bits, ein WORD umfasst 16 Bits, ein DWORD umfasst 32 Bits. Die mit Sternchen (*) markierten Beispielwerte repräsentieren nicht exakt die Feldwerte, da die Felder entweder 6 oder 10 Bits umfassen und die Daten in Vielfachen eines Bytes notiert werden.

Werte, die mehr als ein Byte umfassen, werden im LITTLE ENDIAN-Format oder in umgekehrter Bytereihenfolge gespeichert.

3.2 Dateisysteme erzeugen

Vergleiche die freie Enzyklopädie Wikipedia: http://de.wikipedia.org/wiki/Dateisystem.

Ein Dateisystem ist ein System, das Daten in Form von Dateien auf einem Computersystem speichert und verwaltet. Die meisten Betriebssysteme verwenden Dateisysteme. Historisch wurden Dateisysteme zur Organisation des Zugriffs auf Massenspeicher wie Festplattenlaufwerke entwickelt. Jede Datei belegt einen Teil des Massenspeichers. Ein Dateisystem bietet die Möglichkeit per Namen auf eine Datei zu zugreifen. Das Konzept der Dateisysteme wurde dann soweit abstrahiert, dass auch Zugriffe auf Dateien im Netz und in Geräten, die virtuell als Datei verwaltet werden, über Dateisysteme geregelt werden können.
Dateien haben in einem Dateisystem fast immer mindestens einen Dateinamen sowie Attribute, die nähere Informationen über die Datei geben. Die Dateinamen sind in speziellen Dateien, den Verzeichnissen, abgelegt. Über diese Verzeichnisse kann ein Dateiname und damit eine Datei vom System gefunden werden. Ein Dateisystem bildet somit einen Namensraum. Alle Dateien (oder dateiähnliche Objekte) sind so über einen eindeutigen Namen/Adresse (Dateiname inkl. Pfad oder URL) – innerhalb des Dateisystems – aufrufbar.

```
user@host> man mount
...
       -t vfstype
              The argument following the -t is used to indicate the file  sys-
              tem  type.   The file system types which are currently supported
              are: adfs, affs, autofs, coda, coherent,  cramfs,  devpts,  efs,
              ext,  ext2,  ext3, hfs, hpfs, iso9660, jfs, minix, msdos, ncpfs,
              nfs, ntfs, proc, qnx4,  ramfs,  reiserfs,  romfs,  smbfs,  sysv,
              tmpfs,  udf,  ufs, umsdos, usbfs, vfat, xenix, xfs, xiafs.  Note
              that coherent, sysv and xenix are equivalent and that xenix  and
              coherent will be removed at some point in the future -- use sysv
              instead. Since kernel version 2.1.21 the types ext and xiafs  do
              not exist anymore. Earlier, usbfs was known as usbdevfs.
...
user@host> man filesystems
...
       Below a short description of a few of the available filesystems.

       minix  is the filesystem used in the Minix operating system, the first
              to  run  under  Linux.  It has a number of shortcomings: a 64MB
              partition size limit, short filenames,  a  single  time  stamp,
              etc.  It remains useful for floppies and RAM disks.

       ext    is an elaborate extension of the minix filesystem.  It has been
              completely superseded by the second  version  of  the  extended
              filesystem  (ext2)  and  has  been  removed from the kernel (in
              2.1.21).

       ext2   is the high performance disk filesystem used by Linux for fixed
              disks  as  well  as  removable  media.  The  second extended
              filesystem was designed as an extension of  the  extended  file
              system  (ext).  ext2  offers the best performance (in terms of
              speed and CPU usage) of the filesystems supported under  Linux.

       ext3   is  a  journaling version of the ext2 filesystem. It is easy to
```

switch back and forth between ext2 and ext3. ext3 offers the most complete set of journaling options available among journaling filesystems.

xiafs was designed and implemented to be a stable, safe filesystem by extending the Minix filesystem code. It provides the basic most requested features without undue complexity. The xia filesystem is no longer actively developed or maintained. It was removed from the kernel in 2.1.21.

msdos is the filesystem used by DOS, Windows, and some OS/2 computers. msdos filenames can be no longer than 8 characters, followed by an optional period and 3 character extension.

umsdos is an extended DOS filesystem used by Linux. It adds capability for long filenames, UID/GID, POSIX permissions, and special files (devices, named pipes, etc.) under the DOS filesystem, without sacrificing compatibility with DOS.

vfat is an extended DOS filesystem used by Microsoft Windows95 and Windows NT. VFAT adds the capability to use long filenames under the MSDOS filesystem.

proc is a pseudo-filesystem which is used as an interface to kernel data structures rather than reading and interpreting /dev/kmem. In particular, its files do not take disk space. See proc(5).

iso9660
: is a CD-ROM filesystem type conforming to the ISO 9660 standard.

  High Sierra
  : Linux supports High Sierra, the precursor to the ISO 9660 standard for CD-ROM filesystems. It is automatically recognized within the iso9660 filesystem support under Linux.

  Rock Ridge
  : Linux also supports the System Use Sharing Protocol records specified by the Rock Ridge Interchange Protocol. They are used to further describe the files in the iso9660 filesystem to a UNIX host, and provide information such as long filenames, UID/GID, POSIX permissions, and devices. It is automatically recognized within the iso9660 filesystem support under Linux.

hpfs is the High Performance Filesystem, used in OS/2. This filesystem is read-only under Linux due to the lack of available documentation.

sysv is an implementation of the SystemV/Coherent filesystem for

```
        Linux.  It  implements  all  of  Xenix FS, SystemV/386 FS, and
        Coherent FS.

nfs     is the network filesystem  used  to  access  disks  located  on
        remote computers.

smb     is a network filesystem that supports the SMB protocol, used by
        Windows for Workgroups, Windows NT, and Lan Manager.

        To use smb fs, you need a special mount program, which  can  be
        found      in      the      ksmbfs      package,      found      at
        ftp://sunsite.unc.edu/pub/Linux/system/Filesystems/smbfs.

ncpfs   is a network filesystem that supports the NCP protocol, used by
        Novell NetWare.

        To  use ncpfs, you need special programs, which can be found at
        ftp://linux01.gwdg.de/pub/ncpfs.
```

Siehe auch: http://en.wikipedia.org/wiki/List_of_file_systems und
http://en.wikipedia.org/wiki/Comparison_of_file_systems.

3.2.1 Dateisystem-Erzeugung unter Windows

- Low-level-format/soft sectors: herstellerspezifische Tools, siehe etwa
  http://www.disk-utility.com/hard-disk-low-level-format.html
- Partitionieren unter Windows
- Filesystem-Erzeugung unter Windows

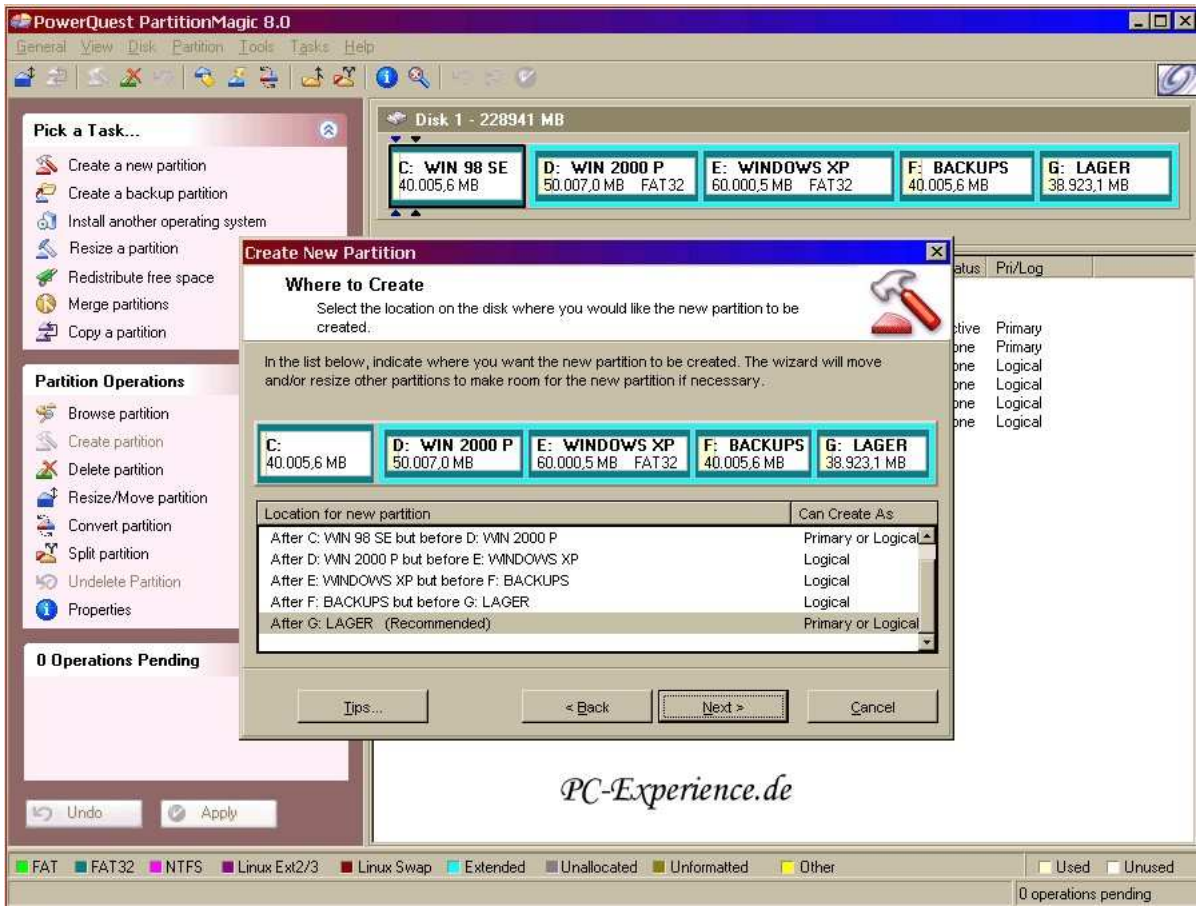


Abbildung 3.4: Partitionierung unter Windows

Die Partitionierung und Filesystemerzeugung geschieht unter Windows entweder bei der Installation des Betriebssystems, mittels des Tools Computerverwaltung (Item Datenträgerverwaltung) beziehungsweise mit speziellen Zusatz-Applikationen wie zum Beispiel PartitionMagic, ...

3.2.2 Dateisystemerzeugung unter Linux

Auch unter Linux werden die Partitionen sowie die Dateisysteme entweder bei der Installation oder mittels des Betriebsystemtools Partitionieren

Abbildung 3.5: Partitionierung unter Linux

erzeugt.

3.2.3 Anlegen eines UNIX-Dateisystems unter Solaris

```
newfs -Nv /dev/rdsk/c0t11d0s7
 s -F ufs -o N /dev/rdsk/c0t11d0s7 275212800 720 8 8192 1024 170 1 167 8192
 v/rdsk/c0t11d0s7:    275212800 sectors in 47780 cylinders of 8 tracks, 720
      134381.2MB in 2655 cyl groups (18 c/g, 50.62MB/g, 6144 i/g)
 er-block backups (for fsck -F ufs -o b=#) at:
 , 104432, 208832, 313232, 417632, 522032, 626432, 730832, 829472, 933872,
 38272, 1142672, 1247072, 1351472, 1455872, 1560272, 1658912, 1763312,
 67712, 1972112, 2076512, 2180912, 2285312, 2389712, 2488352, 2592752,
 97152, 2801552, 2905952, 3010352, 3114752, 3219152, 3317792, 3422192,

 2885792, 272990192, 273094592, 273198992, 273303392, 273407792, 273512192,
 3616592, 273715232, 273819632, 273924032, 274028432, 274132832, 274237232,
 4341632, 274446032, 274544672, 274649072, 274753472, 274857872, 274962272,
 5066672, 275171072,
```

Abbildung 3.6: Anlegen einer Partition mit **newfs**

3.2.4 Zusammenfügen von UNIX-Paritionen: die Mount-Tabelle

```
#device          device          mount           FS      fsck    mount   mount
#to mount        to fsck         point           type    pass    at boot options
#
fd       -       /dev/fd fd      -       no      -
/proc    -       /proc   proc    -       no      -
/dev/dsk/c1t1d0s1        -       -       swap    -       no      -
/dev/dsk/c1t1d0s0        /dev/rdsk/c1t1d0s0      /       ufs     1       no      -
/dev/dsk/c1t2d0s7        /dev/rdsk/c1t2d0s7      /export/home    ufs     2       yes     -
swap     -       /tmp    tmpfs   -       yes     -
```

Abbildung 3.7: Mount-Tabelle

3.2.5 MultiBoot-System: Windows-Partitionen im Boot-Chooser

```
                                    bootini
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(5)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(5)\WINDOWS="Microsoft Windows XP Professional"
/fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional"
/fastdetect
```

Abbildung 3.8: Windows Partitionen

3.3 Dateisystem-Übersicht

3.3.1 Ältere Betriebssysteme

Die älteren Betriebssysteme (CP/M, Apple DOS, Commodore DOS) hatten nur ein Dateisystem, welches keinen eigenen Namen trug. Diese kann man einfach als CP/M-Dateisystem, Apple-Dateisysten, ... bezeichnen.

3.3.2 Neuere Betriebssysteme

- Linux/UNIX:
  - minix (vom gleichnamigen Betriebssystem)
  - ext2 (second extended file system, lange Zeit das Linux-Standard-Dateisystem)
  - ext3 (weiterentwickelte Variante von ext2, mit journaling)
  - FFS (Vorgänger von UFS unter BSD)
  - ReiserFS (Linux Journaling File System von Hans Reiser)
  - JFS (Journaled File System von IBM)
  - UFS (UNIX File System, verwendet unter Solaris und BSD)
  - XFS (ein journaling Dateisystem von SGI)
  - SYSV (das klassische Dateisystem des System V-Unix von AT&T)
  - ADFS (Acorn StrongARM)
  - GNOME Storage (Datenbank-basierendes Dateisystem)
- (MS-)DOS:
  - FAT bzw. FAT12 (File Allocation Table, für Disketten noch heute gebräuchlich)
  - FAT16 (Erweitertes FAT-System für Festplatten)
  - FAT32 (Erweitertes FAT für große Festplatten)
- MS-Windows unterstützt sämtliche DOS-Dateisysteme, zusätzlich:
  - VFAT (Virtual FAT: längere Dateinamen für alle FAT-Systeme)
  - NTFS (Journaling Dateisystem von Windows NT und Nachfolgern)
  - WinFS (für die Zukunft angekündigtes Datenbank-basierendes Dateisystem)
- AmigaOS:
  - OFS (Amiga Old File System)
  - FFS (Amiga Fast File System)
- Apple Macintosh
  - ProDOS (Dateisystem der späten Apple II-Modelle)
  - MFS (Macintosh File System)
  - HFS (Hierarchical File System)
  - HFS+ (Erweiterung von HFS u. a. auf Dateinamen mit mehr als 32 Zeichen)
  - HFSX Case sensitive Variante von HFS+

- OS/2:
  - HPFS (High Performance File System)
  - JFS (Journaled File System)
- BeOS/Haiku:
  - BFS
  - OpenBFS (64 Bit, multithreaded, journaliertes, Datenbank-ähnliches Dateisystem)
- CD-ROM/DVD:
  - ISO9660 (Dateisystem für CD-ROMs)
    - Joliet (Erweiterung des ISO9660 von der Firma Microsoft)
    - Rockridge (Erweiterung des ISO9660 für UNIX)
  - UDF (Universal Disk Format, u. a. auf DVDs aller Typen gebräuchlich)
  - HFS/HFS+ Auf Macintosh-CDs wird häufig das Festplatten-Filesystem genutzt
- Netzwerk:
  - NFS (Network File System; ein über Netzwerke erreichbares Dateisystem für Unix-artige Systeme)
  - Coda (ein fortgeschrittenes Netzwerk-Dateisystem ähnlich zu NFS)
  - SMB (ein über Netzwerke erreichbares Dateisystem vor allem für Windows-Systeme)
  - xFS (ein verteiltes und dezentrales Netzwerk-Dateisystem)
  - AFS (Andrew File System)
  - NCP (NetWare Core Protocol)
  - DFS (distributed file system der Open Group, eine Weiterentwicklung des Andrew File System / Filesystem von Microsoft)

Alle Dateisysteme haben gemeinsam, dass auf sie auch von Fremdsystemen zugegriffen werden kann, sofern das Betriebssystem dies direkt unterstützt oder dies dem Betriebssystem über entsprechende Treibersoftware ermöglicht wird.

Ausnahmen bilden Dateisysteme, die eine erweiterte Berechtigung unterstützen, die Möglichkeit der Verschlüsselung bieten, oder deren genaue Funktionsweise ein Betriebsgeheimnis (proprietär) ist (zum Beispiel NTFS).

3.4 Access Control Lists — ACLs

Zugriffsrechte:
- discretionary
- mandatory

Access Control Lists erlauben die Spezifikation der Zugriffsrechte auf die Objekte des Dateisystems:

- POSIX-ACLs, POSIX-ACLs in Linux
- Windows-ACLs, Windows-AC
- ACLs in heterogenen Windows-/UNIX-Clustern

3.5 Namensräume im Netzwerk / Verteiltes Rechnen

- Newcastle-Connection ...
- Federated Naming Service — fns/xfn
- Samba: Windows-Freigaben

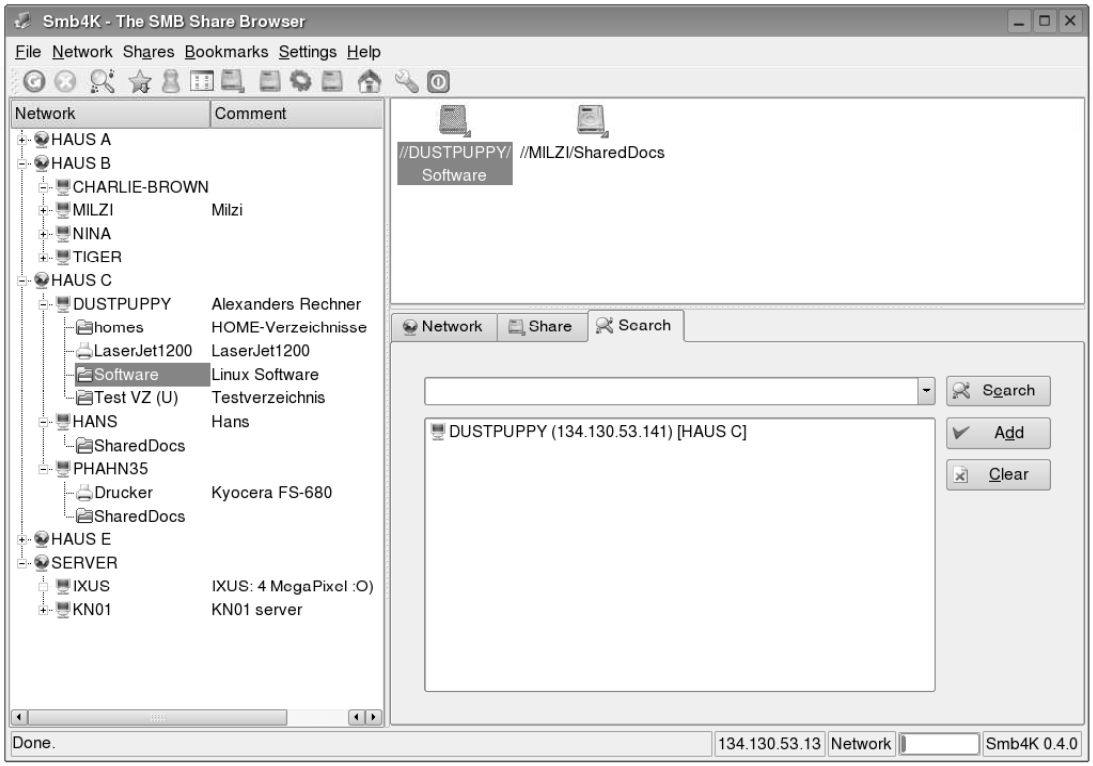


Abbildung 3.9: Windows Shares

- Windows DFS, DFS-Administration
- Service Location Protocol — SLP
- OpenGroup DFS, DCE

3.6 Verteilte Betriebssysteme

Vergleiche: Distributed Computing

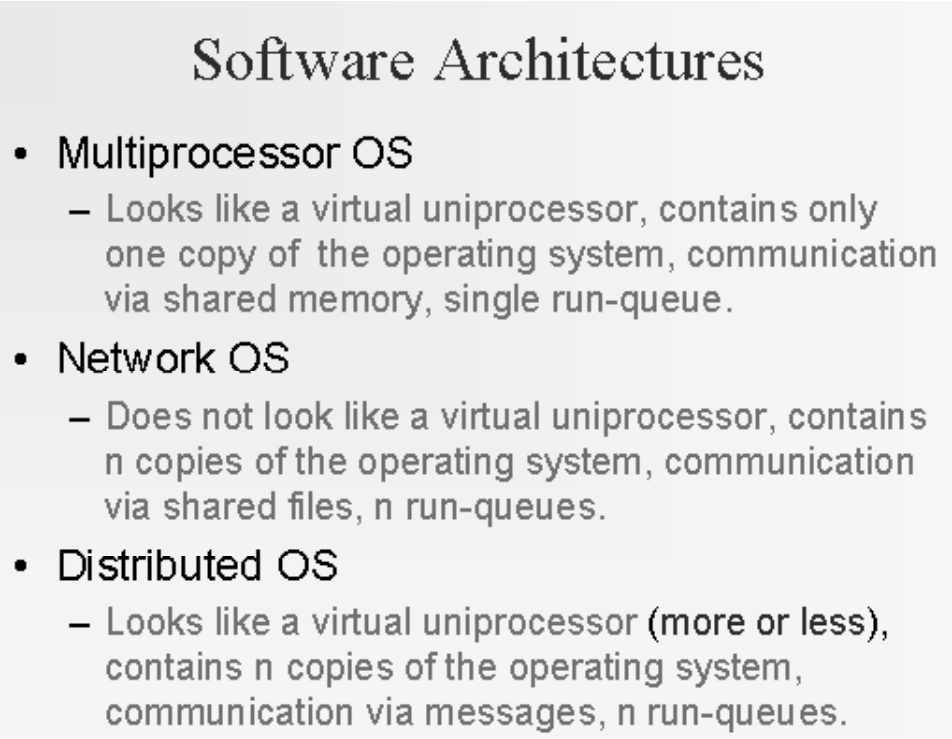


Abbildung 3.10: Verteilte Betriebssysteme

Yair Amir: Distributed Operating Systems

Exemplarisch:

- Amoeba Introduction
- Amoeba User Guide
- Amoeba Administration Guide
- Amoeba Home Page

Weiterführende Links: http://www.informatik.fh-fulda.de/twt/v_sys.htm

3.7 Das FAT-Dateisystem

Die **File Allocation Table** (zu Deutsch: Dateibelegungstabelle) wurde von **Seattle Computer Products** in einem Dateisystem für dessen **Betriebssystem QDOS**, dem direkten Vorgänger von **MS-DOS**, entwickelt. Es ist die einzige bedeutende Neuerung von QDOS gegenüber CP/M. Zu der Familie der FAT-Dateisysteme gehören:

- FAT12 (wird heute immer noch für jede DOS- oder Windows-Diskette gebraucht)
- FAT16 (wird heute für alle Arten von mobilen Datenträgern verwendet, die kleiner als 2 GB sind.)
- FAT32 (wird für alle Arten von mobilen Speichern von mehr als 2 GB Kapazität genutzt)

Die FAT verkettet die Speicher-Cluster der Dateien. Jeder Datei ist im Directory-Eintrag lediglich die Nummer des ersten Clusters des Dateiinhalts (`>=2`) zugeordnet, etwa N. In der FAT steht dann an der Position N die Cluster-Nummer des folgenden Clusters, usw. bis eine spezielle Cluster-Nummer (0xFF8...0xFFF bie FAT12, 0xFFF8...0xFFFF bei FAT16, ...) das Dateiende markiert. Die Cluster-Nummer 0xFF7 (bei FAT12) markiert einen Cluster als unbrauchbar, 0x0 kennzeichnet ihn als frei und 0x1 ist reserviert.
Die **FAT** hat bei den FAT-Dateisystemen in der Regel eine **Kopie**, um bei Datenverlust noch immer eine funktionsfähige alternative FAT zu besitzen. Mit diversen Programmen ist wegen dieser Verkettung der Dateicluster eine Datenwiederherstellung in vielen Fällen möglich, wenn sie früh genug nach dem Datei-Löschbefehl versucht wird.

Vergleiche FAT File System Tutorial
und FAT — How it seems to work.

3.7.1 Aufbau eines FAT-Dateisystems

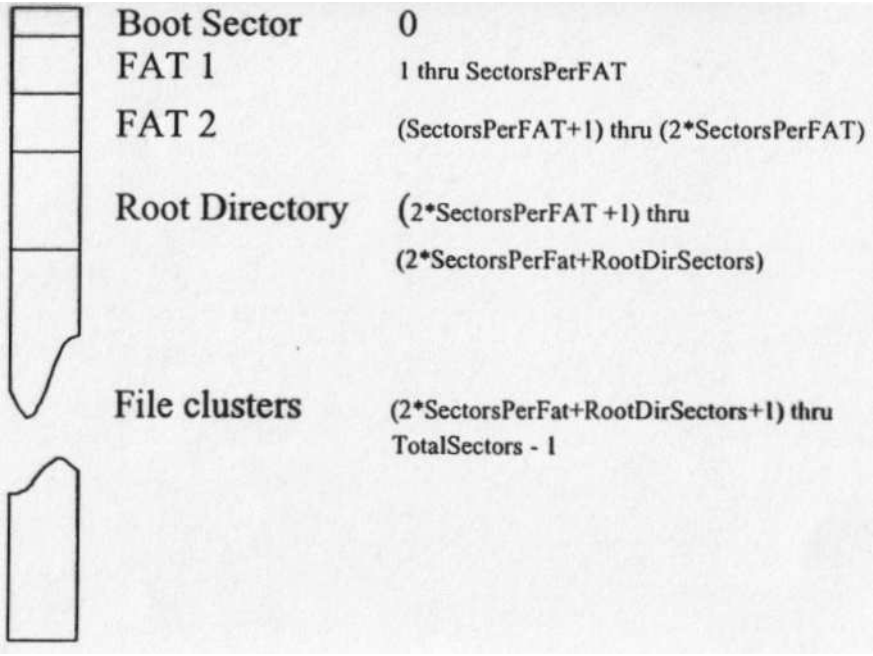


Abbildung 3.11: Struktur und Aufbau einer FAT-Partition

3.7.2 FAT-Verzeichnisse

Inhaltsverzeichnisse/Directories:

- Aneinanderreihung von Verzeichniseinträgen
- Das Wurzelverzeichnis hat eine feste Anzahl von Einträgen
  - dafür sind 14 Sektoren reserviert
- Alle anderen Verzeichnisse sind normale Dateien
  - keine Längen-Beschränkung
- Verzeichniseinträge:
  - 32 Bytes lang
  - das erste Zeichen des Dateinamens ist ein Indikator für:
    - 0x00 Eintrag wurde noch nie benutzt
    - 0xE5 Eintrag wurde benutzt, aber wieder freigegeben (d.h. die Datei wurde gelöscht)
  - Aufbau des Eintrags:

BYTE	Filename[8]	evtl. mit Leerzeichen aufgefüllt
BYTE	Extension[3]	
BYTE	Attribute	
BYTE	Reserviert[10]	
WORD	Zeit	Stunde*2048+Min*32+Sec/2
WORD	Datum	(Jahr-1980)*512 + Monat*32 + Tag
WORD	StartCluster	
DWORD	FileSize	

Die Datei-Attribute:

Bit	Maske	Attribute
0	0x01	Read Only / Leserecht
1	0x02	Hidden / Versteckt Datei
2	0x04	System / Systemdatei
3	0x08	Volume Label / Datenträger-Name
4	0x10	Subdirectory / Unterinhaltsverzeichnis
5	0x20	Archive / Datei ist seit letztem Backup geändert worden
6	0x40	Unused / unbenutzt
7	0x80	Unused / unbenutzt

Tabelle 3.3: File und Datei Attribute

3.8 Das UNIX Filesystem UFS/UFS2

Der UNIX Kern sieht alle Dateien als Byte-Ströme an. Jede innere logische Struktur ist anwendungsspezifisch. Dennoch ist Unix mit physischen Strukturen von verschiedenen Dateiabstraktionen vertraut. Sechs Dateitypen werden unterschieden:

- **Gewöhnlich Dateien:** Dateien, die Informationen von einem Benutzer, einem Anwendungsprogramm oder enem Systemdienstprogramm enthalten.
- **Verzeichnisse:** Dateien, die eine Liste von Verzeichniseinträgen enthalten. Diese enthalten wiederum lediglich den Dateinamen, sowie einen Zeiger zum zugehörigen Inode, der die Datei repräsentiert. Verzeichnisse sind hierarchisch organisiert. Verzeichnis-Dateien sind gewöhnliche Dateien mit speziellem Schreibschutz, so dass nur das Dateisystem in sie schreiben kann, während lesender Zugang für Benutzerprogramme vorhanden ist. Das Root-Verzeichnis des Dateisystems wird durch den ersten Inode realisiert.
- **Spezielle (Geräte-)Dateien:** Sie werden verwendet, um auf Peripheriegeräte wie Terminals und Drucker zuzugreifen. Jeder Eingabe-/Ausgabe-Treiber (I/O-Device) wird mit einer speziellen (Geräte-)Datei angesprochen. Es gibt Block- und Byte-Gerätedateien, die gepufferte bzw. ungepufferte byteweise Ein- und Ausgabe anbieten.
- **Benannte FIFOs:** unidirektionale FIFOs, die durch einen Datei-Pfad im Filesystem erreicht werden können.
- **Symbolische Links:** Durch (symbolische) Indirektionen vergebene weitere Namen für vorhandene Dateien.
- **POSIX IPC-Sockets:** Dateien, die je einen lokalen bidirektionalen Kommunikationsmechanismus für den Datentausch zwischen Prozessen anbieteten.

Vergleiche auch die man-Pages zu fs(5)und newfs(1M).

Maximale UFS Datei- und Dateisystemgrößen:

Betriebssystem	Dateigrösse	Dateisystemgrösse
SunOS 4.1.X	2 GB	2 GB
SunOS 5.X	2 GB	1 TB
IRIX 5.X	2 GB	8 GB
IRIX 6.2	9,000,000 TB (64-bit Kernel) 1 TB (32-bit Kernel)	9,000,000 TB (64-bit Kernel) 1 TB (32-bit Kernel)

Tabelle 3.4: UFS Datei- und Dateisystemgrößen

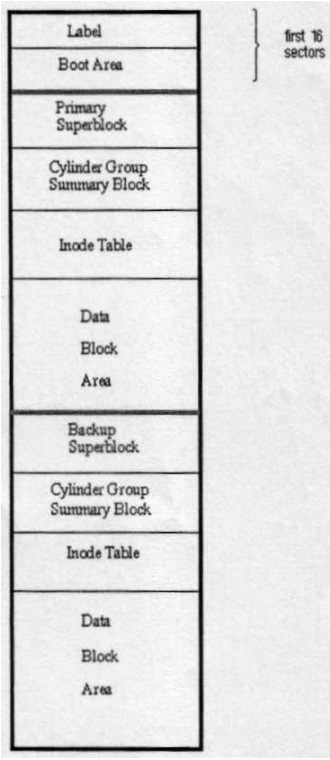


Abbildung 3.12: Aufbau einer UFS-Partition

A UFS filesystem is composed of the following parts:

- a few blocks at the beginning of the partition reserved for boot blocks (which must be initialized separately from the filesystem)
- a superblock, containing a magic number identifying this as a UFS filesystem, and some other vital numbers describing this filesystem's geometry and statistics and behavioral tuning parameters
- a collection of cylinder groups; Each cylinder group has the following components:
  - a backup copy of the superblock
  - a cylinder group header, with statistics, free lists, etc, about this cylinder group, similar to those in the superblock
  - a number of inodes, each containing file attributes
  - a number of data blocks

Inodes are numbered sequentially. The first several inodes are reserved for historical reasons, followed by the inode for the root directory.

Directory files contain only the list of filenames in the directory and the inode associated with each file. All file metadata is kept in the inode.

3.8.1 Inodes

<table>
<tr><th>Feld</th><th>Bedeutung</th></tr>
<tr><td>Datei-Zugriffsrechte</td><td>16-Bit-Flag für die die Zugriffs- und Ausführungsrechte
<table>
<tr><td>12-14</td><td>Dateityp (Datei, Verzeichnis,...)</td></tr>
<tr><td>9-11</td><td>Ausführungs-Flags</td></tr>
<tr><td>8</td><td>Eigentümer-Leseerlaubnis</td></tr>
<tr><td>7</td><td>Eigentümer-Schreiberlaubnis</td></tr>
<tr><td>6</td><td>Eigentümer-Ausführerlaubnis</td></tr>
<tr><td>5</td><td>Gruppen-Leseerlaubnis</td></tr>
<tr><td>4</td><td>Gruppen-Schreiberlaubnis</td></tr>
<tr><td>3</td><td>Gruppen-Ausführerlaubnis</td></tr>
<tr><td>2</td><td>Andere-Leseerlaubnis</td></tr>
<tr><td>1</td><td>Andere-Schreiberlaubnis</td></tr>
<tr><td>0</td><td>Andere-Ausführerlaubnis</td></tr>
</table></td></tr>
<tr><td>Link-Zähler</td><td>Anzahl der Verzeichnissverweise auf diesen Inode</td></tr>
<tr><td>Benutzer-ID</td><td>Eigentümer der Datei</td></tr>
<tr><td>Gruppen-ID</td><td>Eigentümergruppe der Datei</td></tr>
<tr><td>Dateigröße</td><td>Anzahl der Bytes in der Datei</td></tr>
<tr><td>Dateiadressen</td><td>15 zum Teil mehrfach indirekt verwertete Adressen der zur Datei gehörenden Datenblöcke</td></tr>
<tr><td>Letzter Zugriff</td><td>Zeit des letzten Zugriffes</td></tr>
<tr><td>Letzte Modifikation</td><td>Zeit der letzten Modifikation</td></tr>
<tr><td>Letzte Inode-Modifikation</td><td>Zeit der letzten Inode-Modifikation</td></tr>
</table>

Tabelle 3.5: Inode-Aufbau

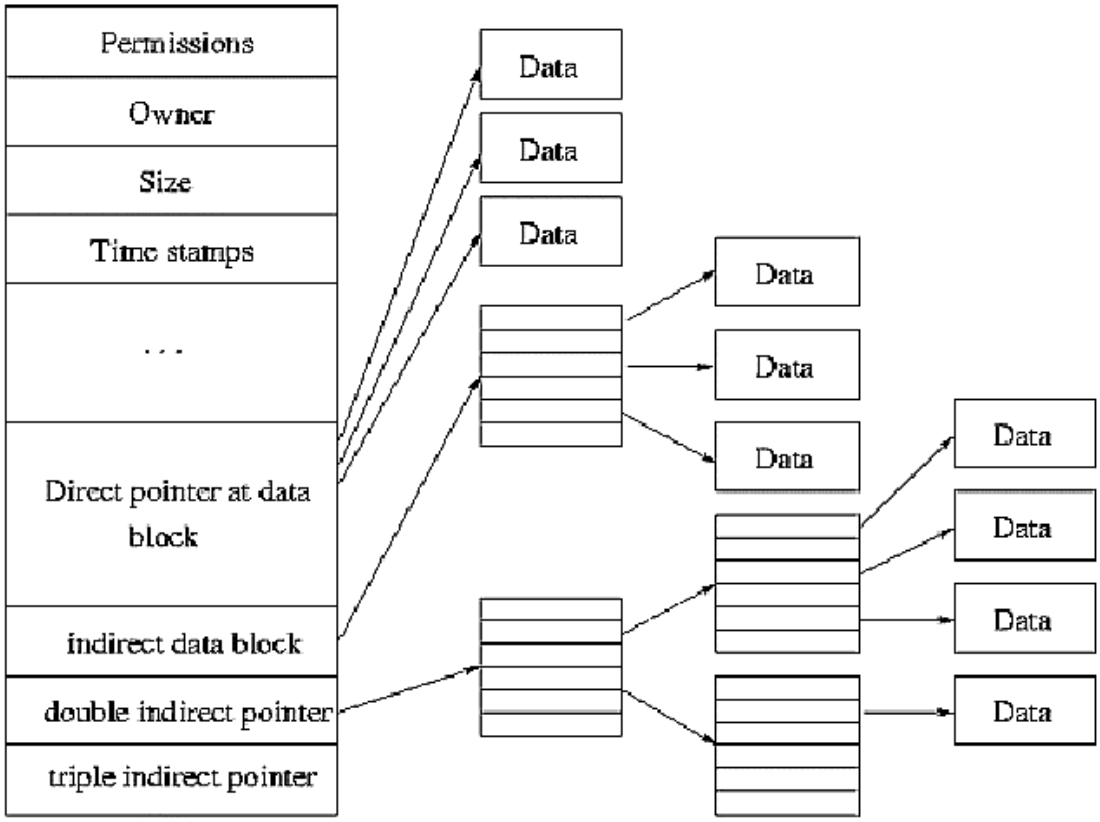


Abbildung 3.13: Adressenschema für die Datenblöcke eines Inodes

Vergleiche: http://neptune.netcomp.monash.edu.au/cpe5013/Lecture/Week03/UFS.html und http://www.sun.com/software/whitepapers/solaris10/fs_performance.pdf.

3.9 Das WINDOWS-Dateisystem NTFS

WINDOWS 2000 (W2K oder WIN 2K) unterstützt mehrere Dateisysteme, einschließlich FAT, das schon auf Windows 95, MS-DOS, und OS/2 lief. Die Entwickler von W2K entwarfen jedoch ein neues Dateisystem, das W2K Dateisystem (NTFS), mit dem beabsichtigt wurde, den hohen Anforderungen eines sicheren Arbeitsplatzes beziehungsweise Servers im kommerziellen Umfeld zu genügen.

Windows NTFS Partition und Cluster Größen:

Volumen Grösse	Sektoren pro Cluster	Clustergrösse
$\leq$ 512 Mbyte	1	512 bytes
512 Mbyte - 1 Gbyte	2	1K
1 Gbyte - 2 Gbyte	4	2K
2 Gbyte - 4 Gbyte	8	4K
4 Gbyte - 8 Gbyte	16	8K
8 Gbyte - 16 Gbyte	32	16K
16 Gbyte - 32 Gbyte	64	32K
$\geq$ 32 Gbyte	128	64K

Eine NTFS-Partition baut sich folgendermaßen auf:

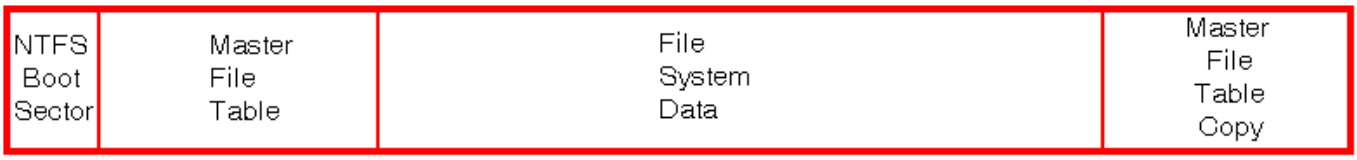


Abbildung 3.14: NTFS-Partition

NTFS-Aufbau
NTFS-Internals

Attribut-Typ	Bedeuteung
Standard-Informationen	Zugriffsrechte (Nur-Lesen, Lesen/Schreiben, ect.), Zeitstempel (Dateierzeugung, Dateimodifikation), Link-Zähler
Attributliste	ergänzt den MFT-Eintrag, wenn dieser nicht genug Platz hat
Filename	eine Datei muß einen oder mehrere Namen besitzen
Sicherheits-Deskriptor	spezifiziert, wem die Datei gehört und wer zugriffsberechtigt ist
Daten(-typ)	Inhalt der Datei. Jede Datei besitzt ein unbenanntes Standard-Datenattribut und darf zusätzliche benannte Attribute besitzen
Index root	für Ordner
Index allocation	für Ordner
Volumen-Informationen	Name und Version des Volumes, ...
Bitmap	Bitmap der belegten MFT-Einträge

Tabelle 3.6: NTFS File- und Verzeichnis-Attribut-Typen

3.9.1 NTFS Schlüssel-Designkonzepte

NTFS ist ein flexibles leistungsstarkes Dateisystem, das auf einem eleganten einfachen Modell basiert. Die beachtenswerten Eigenschaften von NTFS sind unter anderm:

- **Recoverability/Wiederherstellbarkeit:** Ganz oben auf der Liste der Anforderungen an das neue W2K Dateisystem stand die Fähigkeit der Widerherstellbarkeit nach Systemabstürzen und Festplattenfehlern. Im Falle solcher Fehler ist NTFS in der Lage, Plattenvolumina wieder in einen konsistenten Zustand zurückzuversetzen.
  NTFS verwendet ein Transaktions-Modell für Änderungen im Dateisystem: jede signifikante Änderung wird als eine atomare Handlung behandelt, die entweder gänzlich oder überhaupt nicht durchgeführt wird. Jede Transaktion, die zur Zeit eines Fehlers im Gange war, wird beim Recover ungeschehen gemacht oder zur Vollendung gebracht.
  Außerdem verwendet NTFS eine redundante Speicherung für kritische Dateisystem-Daten, so dass ein defekter Plattensektor nicht zum Verlust von Struktur- oder Statusdaten des Dateisystems führen kann.

- **Security/Sicherheit:** NTFS verwendet zur Erzwingung von Sicherhet das W2K Objekt-Modell. Eine offene Datei wird als ein Exemplar eines File-Objekts mit einem Sicherheitsdeskriptor angesehen, der ihre Sicherheitsattribute definiert.

- **Große Platten und große Dateien:** NTFS unterstützt sehr große Festplatten und sehr große Dateien besser und effizienter als die meisten anderen Dateisysteme, FAT eingeschlossen.

- **Multiple data streams/mehrere Datenströme:** Der eigentliche Inhalt einer Datei wird als ein Strom von Bytes behandelt. In NTFS ist es möglich, mehrere Datenströme für eine einzige Datei zu definieren. Ein Beispiel für die Nützlichkeit dieser Eigenschaft ist es, dass NTFS W2K erlaubt, durch entfernte Macintosh-Systeme verwendet zu werden, um Dateien zu speichern und zu laden. Auf einem Macintosh-System hat jede Datei zwei Bestandteile: die Dateidaten und eine Ressourcenbeschreibung, die Information über die Datei enthält. NTFS behandelt diese beiden Komponenten als zwei getrennte Datenströme.

- **General indexing facility/allgemeine Indizierungsmöglichkeit:** NTFS verbindet mit jeder Datei eine Ansammlung von Attributen. Die Menge der Dateibeschreibungen im Dateiverwaltungssystem wird als eine relationale Datenbank organisiert, so dass Dateien nach jedem Attribut indiziert werden können.

3.9.2 Die NTFS Volumen- und Dateistruktur

NTFS verwendet zur Speicherung von Dateinen das folgenge Konzept:

1. **Sektoren:** Die kleinste physikalische Speichereinheit auf der Platte. Die Datengröße ist eine Potenz von 2 und ist fast immer **512 Bytes**
2. **Cluster:** Ein oder mehr aneinander grenzende (nebeneinander in derselben Spur) Sektoren. Die Cluster-Größe in Sektoren ist eine Potenz von 2.
3. **Volumen:** Eine logische Partition einer Festplatte, die aus einem oder mehreren Clustern besteht und durch ein Dateisystem dazu verwendet wird, Plattenplatz zu reservieren. Ein Volumen besteht zu jeder Zeit aus Dateisystem-Informationen, einer Ansammlung von Dateien und zusätzlichem unzugeteilten Plattenplatz, der Dateien zugeteilt werden kann. Ein Volumen kann auf einem Teil einer Platte realisiert sein oder es kann sich über mehrere Platten erstrecken. Wenn ein Hardware- oder Software-RAID 5 verwendet wird, besteht ein Volumen aus Streifen (Stripes), die sich über mehrere Platten verteilen. Die **maximale** Volumen-Größe für NTFS ist 2^{64} Bytes.

Der Cluster ist die fundamentale Zuteilungseinheit in NTFS.

3.9.3 Recoverability/Wiederherstellbarkeit

NTFS ermöglicht es, das Dateisystem nach einem System-Absturz in einen konsistenen Zustand zurückzuführen. Die Schlüsselelemente, die eine Wiederherstellung gewährleisten sind die Folgenden:

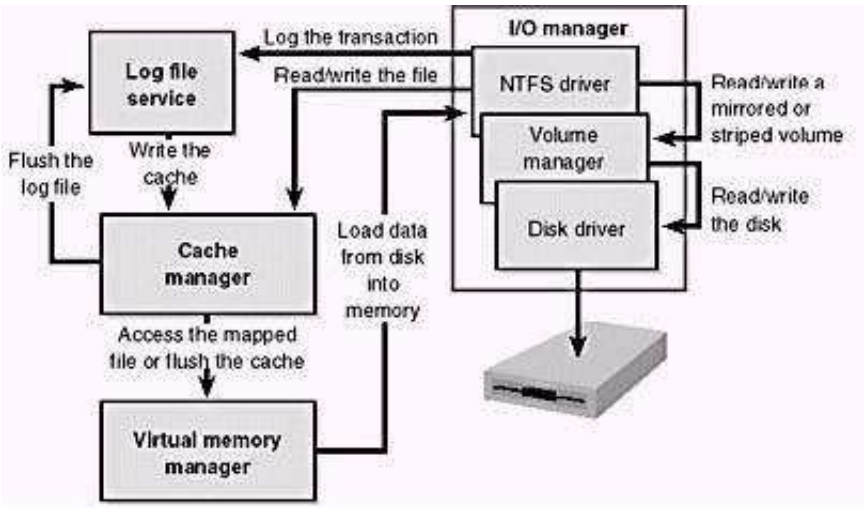


Abbildung 3.15: NTFS-Komponenten

- **I/O manager:** Enthält den NTFS Treiber, der die grundlegenden NTFS-Funktionen Öffnen, Schließen, Lesen, Schreiben von Dateien bereit stellt. Außerdem kann das Software-RAID-Modul FTDISK für die Benutzung konfiguriert werden.

- **Log file service:** Führt ein Log der Platten-Schreiboperationen. Die Log-Datei wird dazu verwendet, ein NTFS-formatiertes Volumen im Falle eines System-Fehlers wiederherzustellen.
- **Cache manager:** Verantwortlich dafür, Datei-Lese- und -Schreib-Operationen zu puffern, um die Leistung zu erhöhen. Der Cache-Manager optimiert das Platten-I/O, indem er eine lazy-write und lazy-commit Technik verwendet.
- **Virtual memory manager:** NTFS greift auf gepufferte Dateien zu, indem es Dateiverweise auf virtuellen Speicher abbildet und virtuellen Speicher liest beziehungsweise schreibt.

Es ist wichtig zu wissen, dass die von NTFS verwendeten Wiederherstellungsverfahren dafür bestimmt sind, Dateisystem-Metadaten wiederherzustellen, nicht Datei-Inhalte. So sollte der Benutzer nie ein Volumen oder die Verzeichniss-/Datei-Struktur einer Applikation wegen eines Absturzes verlieren.

Die folgenden Schritte garantieren die Dateisystem-Wiederherstellung:

1. Zunächst veranlaßt NTFS das Logdateisystem, in der Logdatei im Cache alle Transaktionen zu registrieren, bei der die Volumen-Struktur modifiziert wird.
2. NTFS modifiziert das Volumen (im Cache).
3. Der Cache-Manager weist das Logdateisystem an, die Logdatei auf die Platte zu schreiben.
4. Ist die Logdatei sicher auf der Festplatte gespeichert, schreibt der Cache-Manager die Volumen-Änderungen auf die Platte.

3.10 Weitere UNIX- und LINUX-Dateisysteme

3.10.1 System V FS (S5FS) (1972)

- Erstes UNIX Dateisystem, auch als **FS** bekannt
- Festplatten-Layout besteht aus: **Bootblock**, **Superblock**, **Inodes**, **Datenblöcke**
- **512, 1024, ...** Bytes Blockgröße, **keine Fragmente**
- Sehr einfach und sehr langsam

3.10.2 Berkley Fast File System (FFS, dann UFS/UFS2) (1983, 1994)

- Metadaten werden in **Cylindergruppen** redundant auf der Festplatte verteilt
- Blockgröße ist min. 4 KB, die Fragmentgröße ist 1 KB
- physische Platten-Parameter werden beim Layout berücksichtigt
- Viel bessere Performance
- 256 Byte Dateinamen, Datei-Locks, Symlinks, rename() und Quotas

Verbesserungen:

- 1991: Verbesserungen der Block-Zuteilung und der vorzeitigen Lesepolitik (Larry McVoy und Greg Ganger)
- 1999: Soft updates (Kirk McKusick und Greg Ganger)
- 1999: UFS logging (Neil Perrin)
- 2004: Multi-Terabyte UFS (Lori Alt und Jeff Bonwick)

3.10.3 Log-Structured File System (LFS) (1993)

- Alle **on-disk Daten** inkrementell in Form von aufeinander folgenden Einträgen im zirkularen Log
- Crash-/Fehler-/Störungsbeseitigung vorwärts vom letzten Checkpoint/Kontrollpunkt
- 70% vom gesamten Festplattenplatz effektiv genutzt, aber FFS kann dies ebenso
- Mendel Rosenblum und Jhon K. Ousterhoud, The Design and Implementation of a Log-Structured File System 13th ACM SOSP

3.10.4 ext2 und ext3 (1992, 1999)

- ähnlich wie FFS, keine Fragmente
- ext3 zusätzlich: Journaling System: nach einem Absturz/Crash wird fsck nicht benötigt; H-Tree-Organisation der Verzeichnis-Indices, Dateitypen im Verzeichnis, gelöschte inode-Adressen werden mit Nullen überschrieben
- Primäre Eigenschaften sind **Einfachheit**, **Performanz** und **Wiederhertstellbarkeit**
- vergleiche zur Performanz: http://www.sun.com/software/whitepapers/solaris10/fs_performance.pdf

3.10.5 SGIs XFS (1994)

- FFS mit Extensions zur Performanz-Steigerung
- **B+-Bäume** (Tracks/Spuren) für freien Plattenplatz, Index-Verzeichnisse, lokalisierte Dateiblocks und Inodes; zusätzlich erstes genutztes 64-Bit Dateisystem
- Der Fokus liegt in der Skalierbarkeit und im I/O-Streaming
- Dynamische Inode-Zuteilung, Logging, Volumen-Manager, Multithreadfähig beim lesen/schreiben, ACLs
- online Resizing
- online Defragmentierung
- real-time E/A (Video-Streaming)

3.10.6 IBMs JFS (1990)

- journaling Datgeisystem
- ACLs
- B+Tree Adressierung
- sparse and dense Files

3.10.7 Reiserfs Version 1-4 (ca 2001 - 2005)

- Entworfen für kleine Dateisysteme und große Verzeichniss- Performanz
- online Resizing (groth only), Journaling
- Logging, **B+-Bäume**, tail-Packing
- für kleine Dateien ($\leq$ 4K) 10...15 mal schneller als ext3/ext3
- keine Defragmentierungsmöglichkeiten
- Reiser4: effizienteres Journaling, effizienteres I/O für kleine Dateien, atomic Dateisystem, keine ACLs, Pluggin-Unterstützung (Codierung, Kompression)

3.10.8 Apple Macintosh's HFS+ (1998)

- Journaling
- B-Trees
- Erweitertes jedoch trotzdem abwärtskompatibles HFS+ (z.Zt.: Dateinamen mit signifikanter Groß-/Kleinschreibung)
- Erweiterte Datei-Attribute (Dateityp, ...), Resourcen-Fork
- alternate data stream-Konzept
- ... wird auch für die meisten Macintosh CDs verwendet

3.10.9 Zetabyte File System (ZFS) (2005)

- verfügbar erst in SOLARIS10-Update
- Eingebauter Volumen-Manager: äußerst vereinfachte Volume-Administration (self-Managing)
- 128-Bit Dateisystem
- parallele, O(1)-Verzeichniss-Operationen
- online Resizing
- stets konsistenter Plattenstatus
- Selbstheilende Daten, dynamisches Striping, unterschiedliche Blockgröße, unbegrenzte, zeitkonstante Lese-/Schreib-Snapshots

Vertiefung: http://www.cs.vu.nl/res/theses/oey_thesis.ps

Das Filesystem-Howto

4 RAID — Redundant Arrays of Independent Disks

RAID-Plattenysteme (**Arrays of Inexpensive Disks**) stellen auf der Basis kleiner (billiger) Platten hochperformante und/oder hochsichere virtuelle Platten bereit.
Dem RAID steht das JBOD gegenüber, bei dem zwar mehrere Festplatten innerhalb eines Computers eingebaut werden, aber keine logische Anordnung (Array) entsteht.

4.1 RAID 0: Beschleunigung ohne Fehlertoleranz/Redundanz (engl. striping)

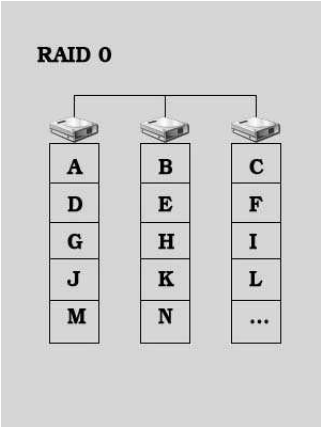


Abbildung 4.1: RAID 0

4.2 RAID 1: Spiegelung (mirroring/duplexing)

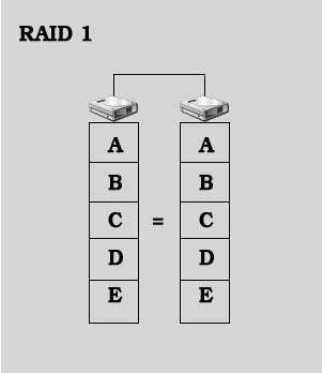


Abbildung 4.2: RAID 1

4.3 RAID 5: Performance + Parität

RAID 5 kombiniert Redundanz mit höherer Performanz:

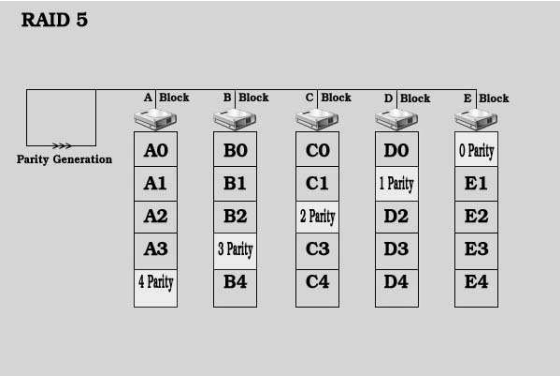


Abbildung 4.3: RAID 5

4.4 RAID 6

RAID 6 funktioniert ähnlich wie RAID 5, verkraftet aber den Ausfall von bis zu zwei Festplatten. Hier werden nicht ein, sondern zwei Fehlerkorrekturwerte berechnet und so über die Platten verteilt, so dass Daten und Paritätsblöcke auf unterschiedlichen Platten liegen.

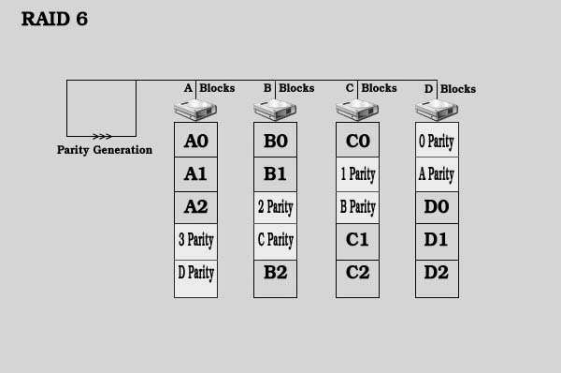


Abbildung 4.4: RAID 6

4.5 Kombinations-RAIDs (RAID 10, 01, ...)

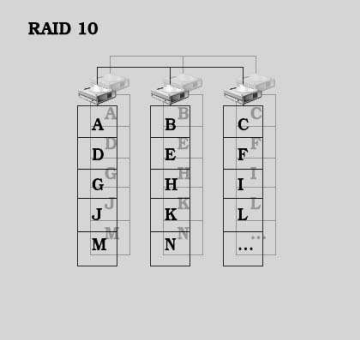


Abbildung 4.5: Kombinations-RAID 10

Weitere Vertiefung:

Fibre-Channel
RAID 01
Hardware-RAID
SW-RAID
Linux LVM

5 Parallel arbeitende Applikationen

5.1 Das Klonen von Prozessen

YoLinux-Tutorial fork()

Prozess-Attribute

Inter-Prozess-Kommunikation

5.2 Starten von (detachten) Threads

YoLinux-Tutorial threads

Multithread Programming

Unix Daemon Server Programming

6 Virtueller Speicher

6.1 Segmentierung

Logische Segmente von Executables: Code, Daten, Stack, Heap, ...

Siehe auch: http://nrg.cs.ucl.ac.uk/mjh/3005/14-segmentation.pdf,
http://www.isg.rhul.ac.uk/msc/teaching/ic4/wk04/hardwaremechanisms.pdf (Seite 86ff.)

6.2 Paging

Siehe: http://www.isg.rhul.ac.uk/msc/teaching/ic4/wk04/hardwaremechanisms.pdf (Seite 88ff.)
http://www.isg.rhul.ac.uk/msc/teaching/ic4/wk04/hardwaremechanisms.pdf (Seite 90ff.)

6.3 Virtueller Speicher

http://en.wikipedia.org/wiki/Virtual_memory

Speicher-Schutz: http://en.wikipedia.org/wiki/Memory_protection

Memory Region Flags

Dax NX-Flag bei Intel-Prozessoren.

7 Sicherheit beim Programmieren

7.1 Sicherheitsaspekte

Informieren Sie sich in „Sichere C-Programmierung“

http://developers.sun.com/solaris/articles/secure.html

über Puffer-Overflows, ...

Richtlinien für Programmierer:

- Überprüfen Sie die Funtionsergebnisse der Systemcalls und beachten Sie den Mißerfolg einer Funktion in der folgenden Programmlogik
- Vermeiden Sie die Benutzung von `system()` und `popen()`; besser ist es, `fork()` und `exec()` zu benutzen, da `system()` beziehungsweise `popen()` eine Shell starten, um das gewünschte Komando auszuführen. Vorsicht ebenso bei `execlp()` und `execvp()`! Jedes Programm sollte im Idealfall eine vorsichtig selbst konstruierte Prozeßumgebung an den Kindprozeß weiterreichen, statt auf die selbst geerbte zu vertrauen!
- Um Datenvertraulichkeit sicherzustellen sollten Sie dafür sorgen, dass der Core-Dump im Programmfehlerfall abgeschaltet ist. (Vergleiche `ulimit(1)` und `setrlimit()`)
- Halten Sie den Programmcode kurz und einfach.
- Stellen Sie sicher, dass Ihr Programmcode Plausibilitätskontrollen (sanity checks) an allen externen Datenquellen durchführt. Alle Funktion (besonders diejenigen, die auf externe Daten angewiesen sind) sollten die Einhaltung von unteren und oberen Schranken für ihre read-Parameter überprüfen. Als Beispiel: gibt es eine Funktion, die nur eine Variable mit einer Größe zwischen 1 und 100 aktieptiert, sollte diese Funktion auch überprüfen, ob diese Variablengrenzen eingehalten werden. Vorsicht auch bei „off by one“-Fehlern!
- Benutzen Sie nur absolute Pfadnamen, insbesondere wenn exec zum Start eines neuen Prozesses benutzt wird.

- Halten Sie sich an den Grundsatz, Algorithmen (-teilen) die geringst möglichen Ausführungs- und Zugriffsrechte zu gewähren, die für die Erfüllung ihrer Aufgaben noch soeben ausreichen!
- Braucht ein Programm höhere Zugriffsrechte, geben Sie diese nicht dem gesamtgen Programm. sondern nur der kleinstmöglichen lokalen Algorithmeneinheit.
- Mit GUI-Programmen ist es unwichtiger geworden, in Programmen Shell-Escapes einzubauen. Verzichten Sie darauf!

7.2 DFN-CERT

Auf den Internet-Seiten des **DFN-CERT** http://www.cert.dfn.de kann man weitere wichtige Informationen zu dem umfassenden Themengebiet **Sicherheit** finden, die Motivation ebenso wie Schwachstellen aufzeigen:

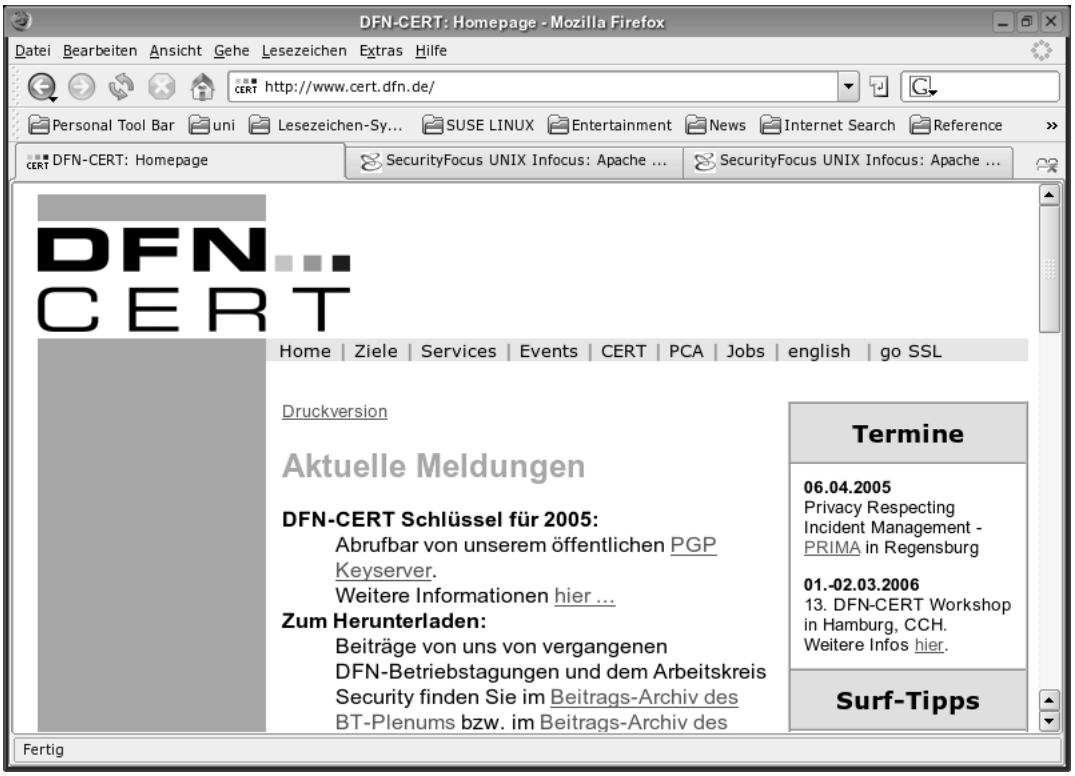


Abbildung 7.1: Internetseite http://www.cert.dfn.de

7.2.1 Puffer-Überläufe und Formatstring-Schwachstellen

Informieren Sie sich in http://www.cert.dfn.de/dfn/berichte/db093/kohlrausch-buffer.pdf über Pufferüberläufe und Formatstring-Schwachstellen in den heutigen Betriebssystemen.

Beherzigen Sie die folgenden Abwehrmaßnahmen:

- Minimale Maßnahmen
  - regelmäßige Einspielung von (Sichnerheits-) Patches
  - Workarounds, Abstellung von besonders fehleranfälligen Optionen (z.B. in Web-Browsern), ...

⇒ Kein Schutz vor noch nicht veröffentlichten oder unbekannten Schwachstellen

⇒ Präventive Maßnahmen notwendig:

- Härten des Systems
- Schutz der SetUID/SetGID Programme
- Schutz/Abstellen der über das Netzwerk angebotenen Dienste

7.3 Präventive Schutzmöglichkeiten für Programmierer

- Sauberes Design (nana, ocl, ...)
- Sicherer Programmierstil
- Manuelle und automatische Analyse des Quelltextes (splint, valgrind, ...)
- Schutz während der Programmausführung (Überwachung der Feldgrenzen, ...)
- Schutz auf Betriebssystemebene
  - willkürlich anordnender (randomizing) Loader
  - Verhindern des Ausführens von Code auf dem Stack und Heap:
    - „non-executable“ Stack ab Solaris 2.6, NX-Flag, ...
    - Linux non-executable Stack
    - PaX für Verhinderung der Codeausführung in beliebigen Speichersegmenten (Stack und Haepsegment)
    - GCC Compiler-Patches StackGuard, Stackshield
    - Absicherung der Rücksprungadresse durch zusätzlichen Code, der vor dem Aufruf bzw. Verlassen einer Funktion ausgeführt wird
    - Vor dem Verlassen der Funktion wird Integrität der Rücksprungadresse überprüft
    - Einfügen eine Canary Wortes direkt vor der Rücksprungadresse im Speicher
      - Annahme: Angreifer kann Canary nicht erraten
      - Exploit überschreibt Canary Wort beim Buffer-Overflow
  - Schutz vor dem Ausnutzen von Format-String Schwachstellen:
    - FormatGuard: glibc-Patch
    - Libsave und Libverify für Linux
      - Libsave: Ersetzen der unsicheren lib Funktion durch sichere Funktionen (z.B. `strcpy`)
      - Libverify: Einfügen von Code zum Schutz der Rücksprungadresse während der Initialisierung des Prozesses (`binary rewriting`)

7.4 Linux gegen Buffer-Overflows schützen

Linux ExecShield

7.5 Buffer-Overflow-Schutz bei Windows XP

NX-Flag
http://www.heise.de/newsticker/meldung/55738

7.6 Secure Linux

SE LKinux Getting Started

Abbildung 7.2: Secure UNIX Programming FAQ, ...