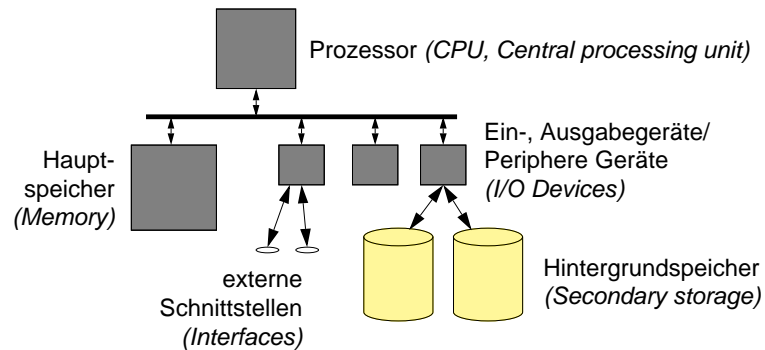


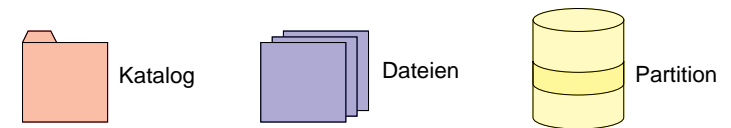
E.1 Allgemeine Konzepte

■ Einordnung



E.1 Allgemeine Konzepte (3)

- Datei
 - ◆ speichert Daten oder Programme
- Katalog / Verzeichnis
 - ◆ erlaubt Benennung der Dateien
 - ◆ enthält Zusatzinformationen zu Dateien
- Partitionen
 - ◆ eine Menge von Katalogen und deren Dateien
 - ◆ Sie dienen zum physischen oder logischen Trennen von Dateimengen.



E.1 Allgemeine Konzepte (2)

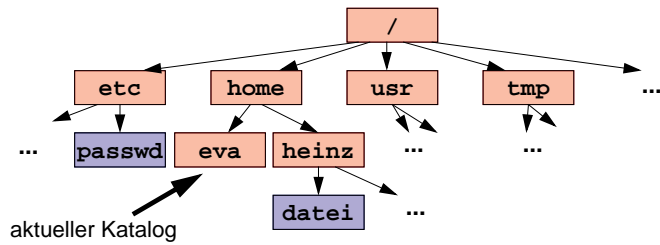
- Dateisysteme speichern Daten und Programme persistent in Dateien
 - ◆ Betriebssystemabstraktion zur Nutzung von Hintergrundspeichern (z.B. Platten, CD-ROM, Floppy Disk, Bandlaufwerke)
 - Benutzer muß sich nicht um die Ansteuerungen verschiedener Speichermedien kümmern
 - einheitliche Sicht auf den Sekundärspeicher
- Dateisysteme bestehen aus
 - ◆ Dateien (*Files*)
 - ◆ Katalogen (*Directories*)
 - ◆ Partitionen (*Partitions*)

E.2 Beispiel: UNIX (Sun-UFS)

- Datei
 - ◆ einfache, unstrukturierte Folge von Bytes
 - ◆ beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
 - ◆ dynamisch erweiterbar
 - ◆ Zugriffsrechte: lesbar, schreibbar, ausführbar
- Katalog
 - ◆ baumförmig strukturiert
 - Knoten des Baums sind Kataloge
 - Blätter des Baums sind Verweise auf Dateien (*Links*)
 - ◆ jedem UNIX Prozeß ist zu jeder Zeit ein aktueller Katalog (*Current working directory*) zugeordnet
 - ◆ Zugriffsrechte: lesbar, schreibbar, durchsuchbar, „nur“ erweiterbar

1 Pfadnamen

Baumstruktur



Pfade

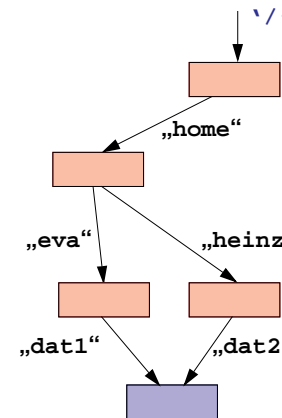
- ◆ z.B. „/home/heinz/datei“, „/tmp“, „../heinz/datei“
- ◆ „/“ ist Trennsymbol (*Slash*); beginnender „/“ bezeichnet Wurzelkatalog; sonst Beginn implizit mit dem aktuellem Katalog

SPL

1 Pfadnamen (3)

Links (Hard links)

- ◆ Dateien können mehrere auf sie zeigende Verweise besitzen, sogenannte Hard links (nicht jedoch Kataloge)

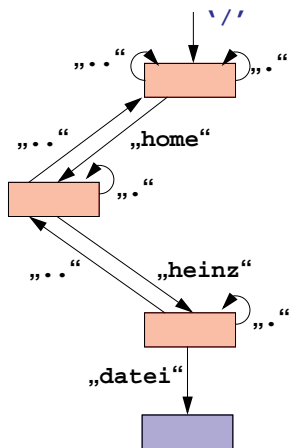


- ◆ Die Datei hat zwei Einträge in verschiedenen Katalogen, die völlig gleichwertig sind:
/home/eva/dat1
/home/heinz/dat2
- ◆ Datei wird erst gelöscht, wenn letzter Link gekappt wird.

SPL

1 Pfadnamen (2)

Eigentliche Baumstruktur



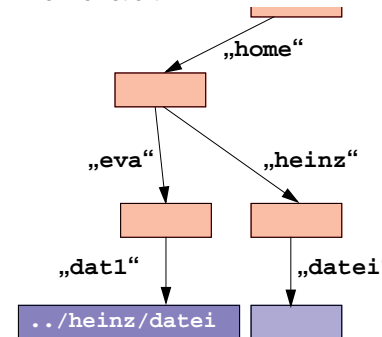
- ▲ benannt sind nicht Dateien und Kataloge, sondern die Verbindungen zwischen ihnen
- ◆ Kataloge und Dateien können auf verschiedenen Pfaden erreichbar sein z.B. ../heinz/datei und /home/heinz/datei
- ◆ Jeder Katalog enthält einen Verweis auf sich selbst („.“) und einen Verweis auf den darüberliegenden Katalog im Baum („..“)

SPL

1 Pfadnamen (4)

Symbolische Namen (Symbolic links)

- ◆ Verweise auf einen anderen Pfadnamen (sowohl auf Dateien als auch Kataloge)
- ◆ Symbolischer Name bleibt auch bestehen, wenn Datei oder Katalog nicht mehr existiert



- ◆ Symbolischer Name enthält einen neuen Pfadnamen, der vom FS interpretiert wird.

SPL

2 Eigentümer und Rechte

- Eigentümer
 - ◆ Jeder Benutzer wird durch eindeutige Nummer (UID) repräsentiert
 - ◆ Ein Benutzer kann einer oder mehreren Benutzergruppen angehören, die durch eine eindeutige Nummer (GID) repräsentiert werden
 - ◆ Eine Datei oder ein Katalog ist genau einem Benutzer und einer Gruppe zugeordnet
- Rechte auf Dateien
 - ◆ Lesen, Schreiben, Ausführen
 - ◆ einzeln für den Eigentümer, für Angehörige der Gruppe und für alle anderen einstellbar
- Rechte auf Kataloge
 - ◆ Lesen, Schreiben (Löschen und Anlegen von Dateien etc.), Durchsuchen
 - ◆ Recht zum Löschen ist einschränkbar auf eigene Dateien

3 Dateien (2)

- Positionieren des Schreib-, Lesezeigers


```
off_t lseek( int fd, off_t offset, int whence );
```
- Attribut-Operationen
 - ◆ Eigentümer und Gruppenzugehörigkeit


```
int chown( char *path, uid_t owner, gid_t group );
```
 - ◆ Zugriffsrechte:

```
int chmod( const char *path, mode_t mode );
```
 - ◆ Länge:

```
int truncate( char *path, off_t length );
```
 - ◆ Zugriffszeiten:

```
int utimes( char *path, struct timeval *tvp );
```
 - ◆ Implizite Maskierung von Rechten:

```
int umask( int mask );
```
- Attribute abfragen


```
int stat( const char *path, struct stat *buf );
```

3 Dateien

- Basisoperationen
 - ◆ Öffnen einer Datei


```
int open( const char *path, int oflag, [mode_t mode] );
```

Rückgabewert ist ein Filedescriptor, mit dem alle weiteren Dateioperationen durchgeführt werden müssen.
 - ◆ Sequentielles Lesen und Schreiben


```
int read( int fd, char *buf, int nbytes );
```

```
int write( int fd, char *buf, int nbytes );
```
 - ◆ Schließen der Datei


```
int close( int fd );
```
- Fehlermeldungen
 - ◆ Anzeige durch Rückgabe von `-1`
 - ◆ Variable `errno` enthält Fehlercode

4 Kataloge

- Kataloge verwalten
 - ◆ Erzeugen


```
int mkdir( const char *path, mode_t mode );
```
 - ◆ Löschen


```
int rmdir( const char *path );
```
 - ◆ Hard link erzeugen


```
int link( const char *existing, const char *new );
```
 - ◆ Symbolischen Namen erzeugen


```
int symlink( const char *path, const char *new );
```
 - ◆ Verweis/Datei löschen


```
int unlink( const char *path );
```

4 Kataloge (2)

■ Kataloge auslesen

- ◆ Öffnen, Lesen und Schließen wie eine normale Datei
- ◆ Interpretation der gelesenen Zeichen ist jedoch systemabhängig, daher wurde eine systemunabhängige Schnittstelle zum Lesen definiert:

```
int getdents(      int fildes, struct dirent *buf,
                  size_t nbyte );
```

- ◆ Zum einfacheren Umgang mit Katalogen gibt es in der Regel Bibliotheksfunktionen:

```
DIR *opendir( const char *path );
struct dirent *readdir( DIR *dirp );
int closedir( DIR *dirp );
```

4 Kataloge (4): readdir

■ Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
```

■ Argumente

u `dirp`: Zeiger auf `DIR`-Datenstruktur

- Rückgabewert: Zeiger auf Datenstruktur vom Typ `struct dirent` oder `NULL` wenn fertig oder Fehler (`errno` vorher auf 0 setzen!)

- Probleme: Der Speicher für `struct dirent` wird von der Bibliothek wieder verwendet!

4 Kataloge (3): opendir / closedir

■ Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);

int closedir(DIR *dirp);
```

■ Argument von opendir

u `dirname`: Verzeichnisname

- Rückgabewert: Zeiger auf Datenstruktur vom Typ `DIR` oder `NULL`

4 Kataloge (5): struct dirent

■ Definition unter Linux (/usr/include/bits/dirent.h)

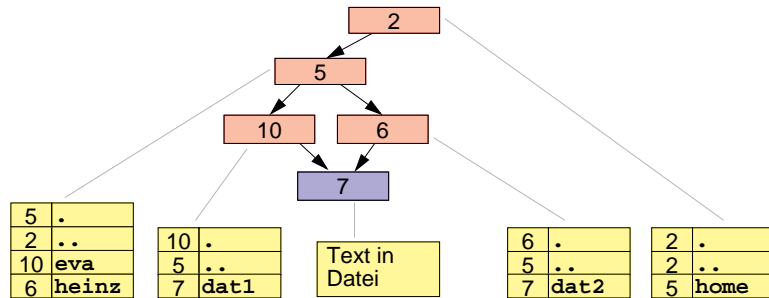
```
struct dirent {
    __ino_t d_ino;
    __off_t d_off;
    unsigned short int d_reclen;
    unsigned char d_type;
    char d_name[256];
};
```

■ Definition unter Solaris (/usr/include/sys/dirent.h)

```
typedef struct dirent {
    ino_t      d_ino;
    off_t      d_off;
    unsigned short d_reclen;
    char       d_name[1];
} dirent_t;
```

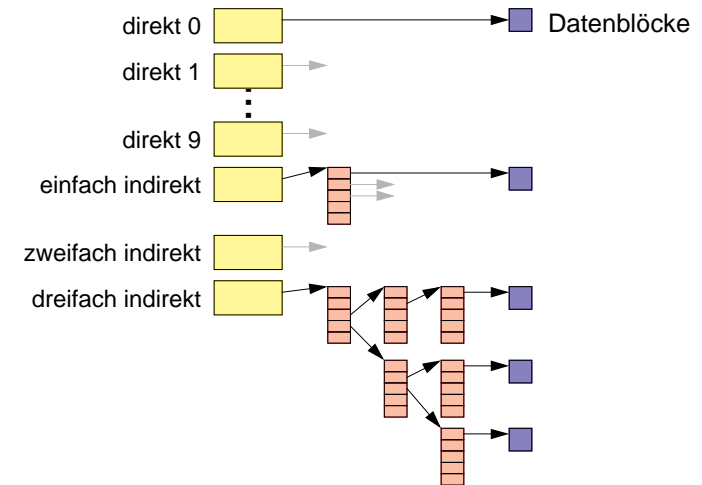
5 Inodes

- Attribute einer Datei und Ortsinformation über ihren Inhalt werden in **Inodes** gehalten
 - ◆ Inodes werden pro Partition numeriert (*Inode number*)
- Kataloge enthalten lediglich Paare von Namen und Inode-Nummern
 - ◆ Kataloge bilden einen hierarchischen Namensraum über einem eigentlich flachen Namensraum (durchnummerierte Dateien)



5 Inodes (3)

- Adressierung der Datenblöcke



5 Inodes (2)

- Inhalt eines Inode
 - ◆ Dateityp: Katalog, normale Datei, Spezialdatei (z.B. Gerät)
 - ◆ Eigentümer und Gruppe
 - ◆ Zugriffsrechte
 - ◆ Zugriffszeiten: letzte Änderung (*mtime*), letzter Zugriff (*atime*), letzte Änderung des Inodes (*ctime*)
 - ◆ Anzahl der Hard links auf den Inode
 - ◆ Dateigröße (in Bytes)
 - ◆ Adressen der Datenblöcke des Datei- oder Kataloginhalts (zehn direkt Adressen und drei indirekte)

6 Spezialdateien

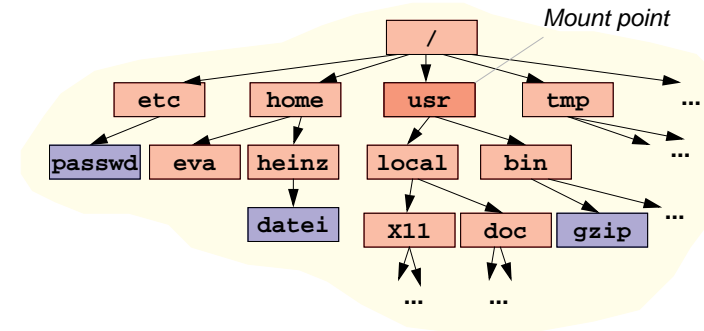
- Periphere Geräte werden als Spezialdateien repräsentiert
 - ◆ Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
 - ◆ Öffnen der Spezialdateien schafft eine (evt. exklusive) Verbindung zum Gerät, die durch einen Treiber hergestellt wird
- Blockorientierte Spezialdateien
 - ◆ Plattenlaufwerke, Bandlaufwerke, Floppy Disks, CD-ROMs
- Zeichenorientierte Spezialdateien
 - ◆ Serielle Schnittstellen, Drucker, Audiokanäle etc.
 - ◆ blockorientierte Geräte haben meist auch eine zusätzliche zeichenorientierte Repräsentation

7 Montieren des Dateibaums

- Der UNIX-Dateibaum kann aus mehreren Partitionen zusammenmontiert werden
 - ◆ Partition wird Dateisystem genannt (*File system*)
 - ◆ wird durch blockorientierte Spezialdatei repräsentiert (z.B. `/dev/dsk/0s3`)
 - ◆ Das Montieren wird *Mounten* genannt
 - ◆ Ausgezeichnetes Dateisystem ist das *Root file system*, dessen Wurzelkatalog gleichzeitig Wurzelkatalog des Gesamtsystems ist
 - ◆ Andere Dateisysteme können mit dem Befehl `mount` in das bestehende System hineinmontiert werden

7 Montieren des Dateibaums (2)

- Beispiel nach Ausführung des Montierbefehls



7 Montieren des Dateibaums (2)

- Beispiel

