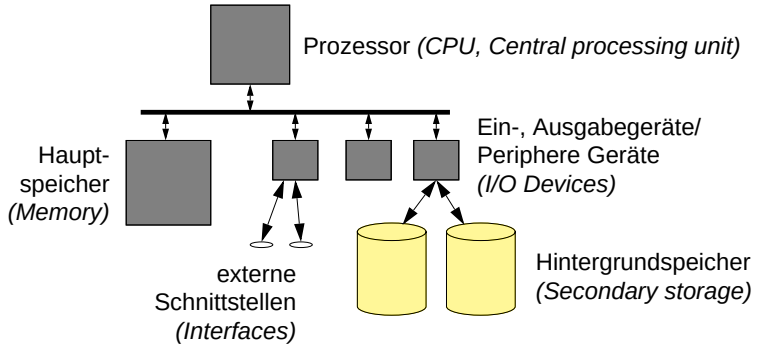
Überblick: Teil D Betriebssystemabstraktionen

Überblick

- Einordnung

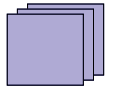


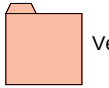
Überblick (2)

- Dateisysteme speichern Daten und Programme persistent in Dateien
  - Betriebssystemabstraktion zur Nutzung von Hintergrundspeicher (z. B. Platten, SSD / Flash-Speicher, DVD / CD-ROM, Bandlaufwerke)
    - Benutzer muss sich nicht um die Ansteuerungen verschiedener Speichermedien kümmern
    - einheitliche Sicht auf den Hintergrundspeicher
- Wesentliche Elemente eines Dateisystems:
  - Dateien *(Files)*
  - Verzeichnissen / Katalogen *(Directories)*
  - Partitionen *(Partitions)*

Überblick (3)

- Datei
  - speichert Daten oder Programme
- Verzeichnis / Katalog (*Directory*)
  - fasst Dateien (u. Verzeichnisse) zusammen
  - erlaubt Benennung der Dateien
  - ermöglicht Aufbau eines hierarchischen Namensraums
  - enthält Zusatzinformationen zu Dateien
- Partition
  - eine Menge von Verzeichnissen und deren Dateien
  - sie dienen zum physischen oder logischen Trennen von Dateimengen
    - ➤ *physisch:* Festplatte, Diskette
    - ➤ *logisch:* Teilbereich auf Platte oder CD





Dateien

- Kleinste Einheit, in der etwas auf den Hintergrundspeicher geschrieben werden kann.

Dateiattribute

- *Name* — Symbolischer Name, vom Benutzer les- und interpretierbar
  - z. B. **`AUTOEXEC.BAT`**
- *Typ* — Für Dateisysteme, die verschiedene Dateitypen unterscheiden
  - z. B. sequenzielle Datei, zeichenorientierte Datei, satzorientierte Datei
- *Ortsinformation* — Wo werden die Daten physisch gespeichert?
  - Gerätenummer, Nummern der Plattenblocks

Dateiattribute (2)

- *Größe* — Länge der Datei in Größeneinheiten (z. B. Bytes, Blöcke, Sätze)
  - steht in engem Zusammenhang mit der Ortsinformation
  - wird zum Prüfen der Dateigrenzen z. B. beim Lesen benötigt
- *Zeitstempel* — z. B. Zeit und Datum der Erstellung, letzten Änderung
  - unterstützt Backup, Entwicklungswerkzeuge, Benutzerüberwachung etc.
- *Rechte* — Zugriffsrechte, z. B. Lese-, Schreibberechtigung
  - z. B. nur für den Eigentümer schreibbar, für alle anderen nur lesbar
- *Eigentümer* — Identifikation des Eigentümers
  - eventuell eng mit den Rechten verknüpft
  - Zuordnung beim Accounting (Abrechnung des Plattenplatzes)

Operationen auf Dateien

- Erzeugen *(Create)*
  - Nötiger Speicherplatz wird angefordert.
  - Verzeichniseintrag wird erstellt.
  - Initiale Attribute werden gespeichert.
- Schreiben *(Write)*
  - Identifikation der Datei
  - Daten werden auf Platte transferiert.
  - eventuell Nachfordern von Speicherplatz
  - eventuelle Anpassung der Attribute, z. B. Länge, Zugriffszeit
- Lesen *(Read)*
  - Identifikation der Datei
  - Daten werden von Platte gelesen.

Operationen auf Dateien (2)

- Positionieren des Schreib-/Lesezeigers für die nächste Schreib- oder Leseoperation *(Seek)*
  - Identifikation der Datei
  - In vielen Systemen wird dieser Zeiger implizit bei Schreib- und Leseoperationen positioniert.
  - Ermöglicht explizites Positionieren
- Verkürzen *(Truncate)*
  - Identifikation der Datei
  - Ab einer bestimmten Position wird der Inhalt entfernt (evtl. kann nur der Gesamtinhalt gelöscht werden).
  - Anpassung der betroffenen Attribute
- Löschen *(Delete)*
  - Identifikation der Datei
  - Entfernen der Datei aus dem Verzeichnis und Freigabe der Plattenblöcke

Verzeichnisse / Kataloge

- Ein Verzeichnis gruppiert Dateien und evtl. weitere Verzeichnisse
- Gruppierungsalternativen:
  - Verknüpfung mit der Benennung
    - Verzeichnis enthält Namen und Verweise auf Dateien und andere Verzeichnisse, z. B. *UNIX, Windows*
  - Gruppierung über Bedingung
    - Verzeichnis enthält Namen und Verweise auf Dateien, die einer bestimmten Bedingung gehorchen
      z. B. gleiche Gruppennummer in *CP/M*
      z. B. eigenschaftsorientierte und dynamische Gruppierung in *BeOS-BFS*
- Verzeichnis ermöglicht das Auffinden von Dateien
  - Vermittlung zwischen externer und interner Bezeichnung (Dateiname — Plattenblöcke)

Operationen auf Verzeichnissen

- Auslesen der Einträge *(Read, Read Directory)*
  - Daten des Verzeichnisinhalts werden gelesen und meist eintragsweise zurückgegeben
- Erzeugen und Löschen der Einträge erfolgt implizit mit der zugehörigen Dateioperation
- Erzeugen und Löschen von Verzeichnissen *(Create and Delete Directory)*

Attribute von Verzeichnissen

- Die meisten Dateiattribute treffen auch für Verzeichnisse zu
  - Name, Ortsinformationen, Größe, Zeitstempel, Rechte, Eigentümer

Dateisystem am Beispiel Linux/UNIX

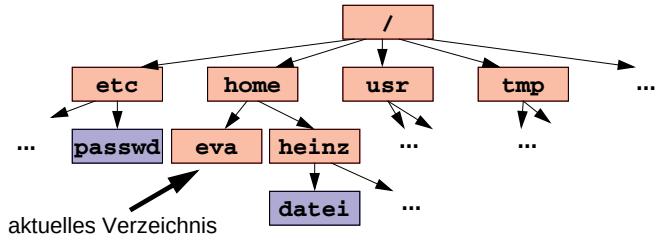
- Datei
  - einfache, unstrukturierte Folge von Bytes
  - beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
  - dynamisch erweiterbar
- Dateiattribute
  - das Betriebssystem verwaltet zu jeder Datei eine Reihe von Attributen (Rechte, Größe, Zugriffszeiten, Adressen der Datenblöcke, …)
  - die Attribute werden in einer speziellen Verwaltungsstrukur, dem *Dateikopf*, gespeichert
    - ➤ Linux/UNIX: *Inode*
    - ➤ Windows NTFS: *Master File Table*-Eintrag
- Namensraum
  - Flacher Namensraum: Inodes sind einfach durchnummeriert
  - Hierarchischer Namensraum: Verzeichnisstruktur bildet Datei- und Pfadnamen in einem Dateibaum auf Inode-Nummern ab

Namensräume

- Ein Namensraum definiert den Kontext, in dem ein Name eine Bedeutung hat ("Java" im Kontext Geografie, Programmiersprachen; "C" im Kontext Musik, Alphabet, Programmiersprachen)
- Flache Namensräume
  - jeder Name muss eindeutig sein
  - bei einer großen Menge von Elementen unübersichtlich und schwer zu verwalten
- Hierarchische Namensräume
  - Menge von Kontexten, die irgendwelche zu benennenden Elemente, auch Kontexte, enthalten
  - am Beispiel UNIX: baumförmig strukturierter hierarchischer Namensraum
    - Knoten (Kontexte) des Baums sind Verzeichnisse (*Directories*)
    - Blätter des Baums sind Verweise auf Dateien
  - jedem UNIX-Prozess ist zu jeder Zeit ein aktuelles Verzeichnis *(Current working directory)* zugeordnet

Pfadnamen

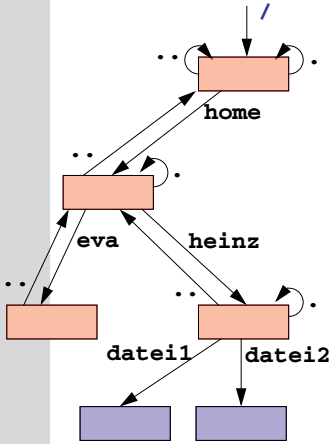
- Baumstruktur



- Pfade
  - z. B. „**`/home/heinz/datei`**“, „**`/tmp`**“, „**`../heinz/datei`**“
  - „**`/`**“ ist Trennsymbol *(Slash)*; beginnender „**`/`**“ bezeichnet Wurzelverzeichnis; sonst Beginn implizit mit dem aktuellen Verzeichnis

Pfadnamen (2)

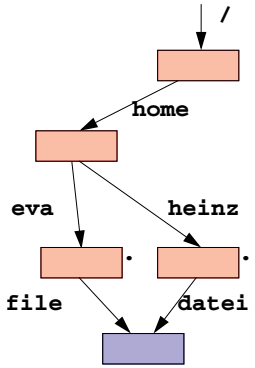
- Eigentliche Baumstruktur



- benannt sind nicht Dateien und Verzeichnisse, sondern die Verbindungen (*Links*) zwischen ihnen
  - Verzeichnisse und Dateien können auf verschiedenen Pfaden erreichbar sein
    z. B. **`../heinz/datei1`** und **`/home/heinz/datei1`**
  - Jedes Verzeichnis enthält
    - einen Verweis auf sich selbst (**`.`**) und
    - einen Verweis auf das darüberliegende Verzeichnis im Baum (**`..`**)
    - Verweise auf Dateien

Pfadnamen (3)

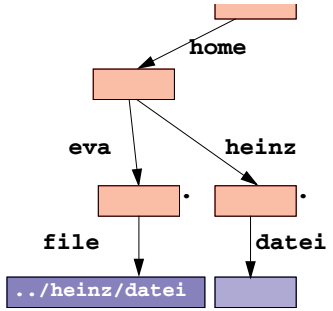
- Links *(Hard Links)*
  - Dateien können mehrere auf sich zeigende Verweise besitzen, sogenannte Hard-Links (nicht jedoch Verzeichnisse)



  - Die Datei hat zwei Einträge in verschiedenen Verzeichnissen, die völlig gleichwertig sind:
    `/home/eva/file`
    `/home/heinz/datei`
  - Datei wird erst gelöscht, wenn letzter Link gekappt wird.

Pfadnamen (4)

- Symbolische Namen *(Symbolic Links)*
  - Verweise auf einen anderen Pfadnamen (sowohl auf Dateien als auch Verzeichnisse)
  - Symbolischer Name bleibt auch bestehen, wenn Datei oder Verzeichnis nicht mehr existiert



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

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 Verzeichnis ist genau einem Benutzer und einer Gruppe zugeordnet
- Rechte auf Dateien
  - Lesen, Schreiben, Ausführen (nur vom Eigentümer veränderbar)
  - einzeln für den Eigentümer, für Angehörige der Gruppe und für alle anderen einstellbar
- Rechte auf Verzeichnissen
  - Lesen, Schreiben (Löschen u. Anlegen von Dateien etc.), Durchgangsrecht
  - Schreibrecht ist einschränkbar auf eigene Dateien („nur erweiterbarer“)

Inodes

- Attribute (Zugriffsrechte, Eigentümer, etc.) einer Datei und Ortsinformation über ihren Inhalt werden in **Inodes** gehalten
  - Inodes werden pro Partition nummeriert *(Inode number)*
- Inhalt eines Inode: Dateiattribute
  - Dateityp: Verzeichnis, normale Datei, Spezialdatei (z. B. Gerät)
  - Eigentümer und Gruppe
  - Zugriffsrechte
  - Zugriffszeiten: letzte Änderung *(mtime)*, letzer Zugriff *(atime)*, letzte Änderung des Inodes *(ctime)*
  - Anzahl der Hard links auf den Inode
  - Dateigröße (in Bytes)
  - Gerät (Disk, Partition), auf dem der Inode angelegt ist
  - Adressen der Datenblöcke des Dateiinhalts
  - bei Spezialdateien: Typ und laufende Nummer des zugeordneten Geräts

Inodes — Programmierschnittstelle: stat / lstat

- liefert Dateiattribute aus dem Inode
- Funktionsschnittstelle:

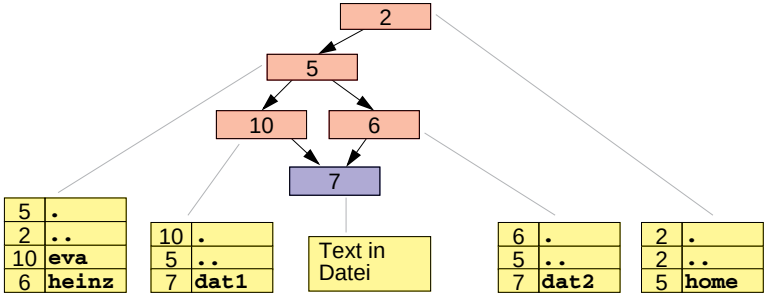
```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *path, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

- Argumente:
  - **`path`**: Dateiname
  - **`buf`**: Zeiger auf Puffer, in den Inode-Informationen eingetragen werden
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel:

```
struct stat buf;
stat("/etc/passwd", &buf); /* Fehlerabfrage ... */
printf("Inode-Nummer: %d\n", buf.st_ino);
```


Verzeichnisse, Pfadnamen und Inodes

- Verzeichnisse enthalten lediglich Paare von Namen und Inode-Nummern
  - Verzeichnisse bilden einen hierarchischen Namensraum über einem eigentlich flachen Namensraum (durchnumerierte Dateien)
  - Die Verzeichnisstrukturen mit ihren Funktionen agieren damit als *Nameserver*, der Pfadnamen in Inode-Nummern abbildet



- Dateiattribute stehen im Inode, nicht im Verzeichnis! (oft benötigte Attribute, z. B. Dateigröße, können aber zusätzlich im Verzeichnis gehalten werden)

Programmierschnittstelle für Verzeichnisse

- Verzeichnisse 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 );
```

- Symbolische Namen auslesen

```
int readlink( const char *path, void *buf, size_t bufsiz );
```


Programmierschnittstelle für Verzeichnisse (2)

- Verzeichnisse lesen (Schnittstelle der C-Bibliothek)
  - Verzeichnis öffnen:
    `DIR *opendir( const char *path );`
  - Verzeichniseinträge lesen:
    `struct dirent *readdir( DIR *dirp );`
  - Verzeichnis schließen:
    `int closedir( DIR *dirp );`

Verzeichnisse (2): opendir / closedir

- Funktionsschnittstelle:

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

DIR *opendir(const char *dirname);

int closedir(DIR *dirp);
```

- Argument von opendir
  - **`dirname`**: Verzeichnisname
- Rückgabewert: Zeiger auf Datenstruktur vom Typ **`DIR`** oder **`NULL`**

Verzeichnisse (3): readdir

- Funktionsschnittstelle:

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

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

- Argumente
  - **`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 Funktion readdir beim nächsten Aufruf wieder verwendet!
  - wenn Daten aus der Struktur (z. B. der Dateiname) länger benötigt werden, reicht es nicht, sich den zurückgegebenen Zeiger zu merken sondern es müssen die benötigten Daten kopiert werden

Verzeichnisse (4): 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];
};
```


Programmierschnittstelle für Dateien

- siehe C-Ein/Ausgabe (Schnittstelle der C-Bibliothek)
- C-Funktionen (fopen, printf, scanf, getchar, fputs, fclose, ...) verbergen die "eigentliche" Systemschnittstelle und bieten mehr "Komfort"
  - ➤ Systemschnittstelle: open, close, read, write

Spezialdateien

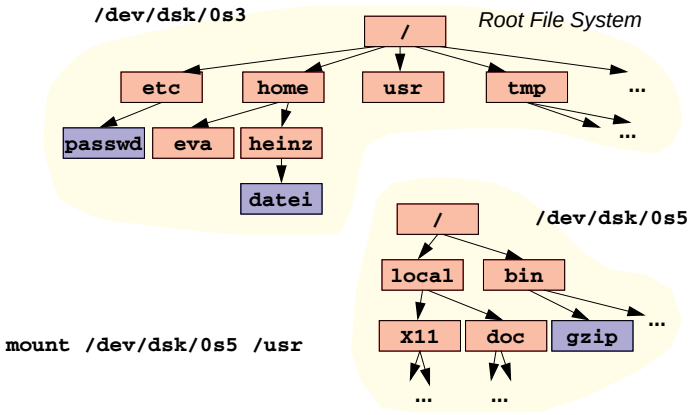
- Periphere Geräte werden als Spezialdateien repräsentiert
  - Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
  - Öffnen der Spezialdateien schafft eine (evtl. exklusive) Verbindung zum Gerät, die durch einen Treiber hergestellt wird
- Blockorientierte Spezialdateien
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

Partitionen

- jede Partition enthält einen eigenen Dateibaum
  (= "ein eigenes Dateisystem")
  - wird durch blockorientierte Spezialdatei repräsentiert
    (z. B. **`/dev/dsk/0s3`**)
- Bäume der Partitionen können zu einem homogenen Dateibaum zusammengebaut werden (Grenzen für Anwender nicht sichtbar!)
  - "Montieren" von Dateibäumen (*mounting*)
- Ein ausgezeichnetes Dateisystem ist das *Root File System*, dessen Wurzelverzeichnis gleichzeitig Wurzelverzeichnis des Gesamtsystems ist
  - Andere Dateisysteme können mit dem Befehl **`mount`** in das bestehende System hineinmontiert werden
  - Über *Network File System* (NFS) können auch Verzeichnisse anderer Rechner in einen lokalen Dateibaum hineinmontiert werden
    ➥ Grenzen zwischen Dateisystemen verschiedener Rechner weren unsichtbar

Montieren des Dateibaums

- Beispiel



Montieren des Dateibaums (2)

- Beispiel nach Ausführung des Montierbefehls