J UNIX-Dateisystem

J.1 Funktionalität

- Abstraktionen für Benutzersicht:
  1. Pfade, Dateinamen
  2. Dateibaum verdeckt mehrere Platten (bzw. Partitionen)
  3. Dateien = unstrukturierte Byteströme
  4. *special files* fast wie reguläre Dateien zugreifbar
- diese Abstraktionen müssen vom Dateisystem auf konkrete Objekte abgebildet werden
- den obigen Abstraktionen entsprechen etwa folgende Objekte:
  - 1. / 2. Plattenpositionen, Adressen von Gerätecontrollern, etc.
  - 3. Platten verwalten Sektoren, Spuren und Zylinder
  - 4. unterschiedliche Gerätetreiber werden für E/A-Operationen auf verschiedenen Peripheriegeräten benötigt

J.2 Directories (Kataloge)

- wichtigster Bestandteil zum Aufbau eines hierarchischen Dateisystems
- Funktion: Strukturierung des Namensraums
- *Directories* enthalten Verweise auf Inodes (von Dateien oder weiteren Directories)

J.3 Inodes

- ein Inode enthält alle Informationen, die zur Verwaltung einer Datei und zum Auffinden ihres Inhalts benötigt werden
- beim Öffnen einer Datei werden die Inode-Daten von der Platte ***(disk inode)*** in die Inode-Tabelle des Systemkerns (***in-core inode)*** gelesen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Umwandlung: Pfad : Inode

- Root = Inode 2
- Inode-Nummer ist nur in dem Directory zu finden, in dem die Datei eingetragen ist
  - ➥ die Pfadkomponenten müssen einzeln bearbeitet werden
  - ➥ um den Inode der jeweils nächsten Pfadkomponente zu bestimmen, muß der Inhalt des voranstehenden Directories eingelesen werden
- Systemkern-Funktion: ***lookup( )***
- Directory Lookup Cache zur Beschleunigung
  - ◆ LRU Cache mit Einträgen
    Directory-Inode-Nummer / Datei(name) in Directory / Inode der Datei

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Umwandlung: Pfad : Inode (2)

- Berücksichtigt werden hierbei:
  - ◆ Symbolic Links
    - – werden beim Absteigen im Baum verfolgt
  - ◆ Mount-Punkte
    - – beim Absteigen im Baum am Mount-Punkt Wechsel zur *root* des eingehängten Dateisystems
    - – beim Aufsteigen im Baum (über "..") an der Root eines Dateisystems Wechsel auf den Mount-Punkt
  - ◆ Current Root
    - – Überschreiten beim Aufsteigen im Baum (über "..") wird verhindert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen

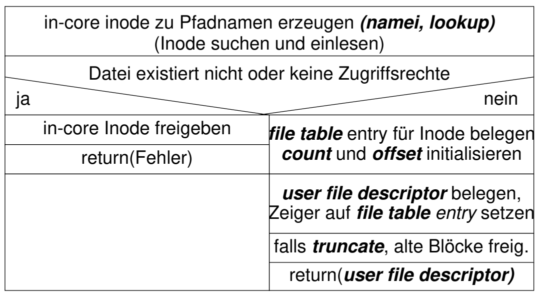
- Aufruf:

```
fd = open(path, flags, modes);
```

fd	Filedescriptor (kleiner Integer-Wert)
path	Pfadname der Datei
flags	Angabe über Zugriffsform (lesen, schreiben, lesen+schreiben), ob Datei neu erzeugt / vorher gelöscht wird
modes	Zugriffsrechte, falls die Datei neu erzeugt wird

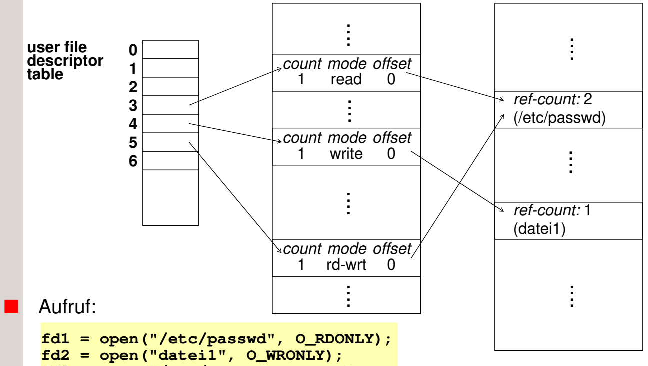
2 Datei öffnen (2)

- open-Algorithmus



2 Datei öffnen (3)

- Kontrollstrukturen (früher)



- Aufruf:

```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);

-> fd1 == 3, fd2 == 4, fd3 == 5
```


2 Datei öffnen (4)

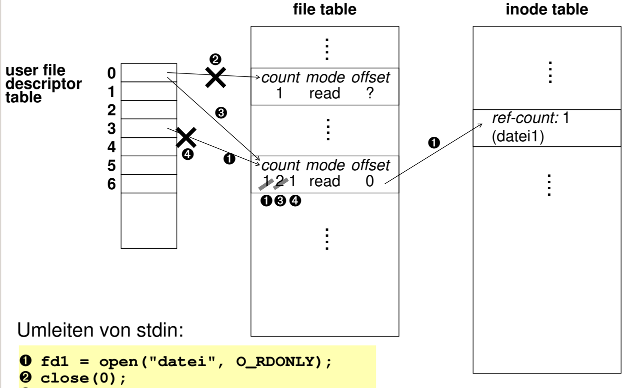
- **user file descriptor table**
  - Feld von Zeigern auf Einträge in der ***file table***
  - Teil der ***user-area*** des Prozesses
    `struct file *u_ofile[NOFILE]`
    siehe *<sys/user.h>*
  - Filedescriptor (aus ***open(2)***) ist ein Index in diese Tabelle
  - ➥ Entkopplung von Systemdatenstrukturen und Benutzerdaten
    Filedeskriptor = Capability
    (Sicherheit!)

- **file table**
  - Feld von ***file***-Strukturen (siehe *<sys/file.h>*), enthält
    - ➤ Zeiger auf Eintrag in der ***inode table***
    - ➤ Eintrag für Zugriffsmodus
    - ➤ Schreib/Lesezeiger (offset)
    - ➤ Referenzzähler

2 Datei öffnen (5)

- **inode table**
  - ◆ Feld von ***in-core inodes*** (siehe *<ufs/inode.h>*), enthält
    - ➤ Referenzzähler
    - ➤ Info über physikalische Gerät (Platte, Partition)
    - ➤ *disk inode* Struktur
    - ➤ Informationen für ***lock***-Funktionen (***flock(2), lockf(2)***)

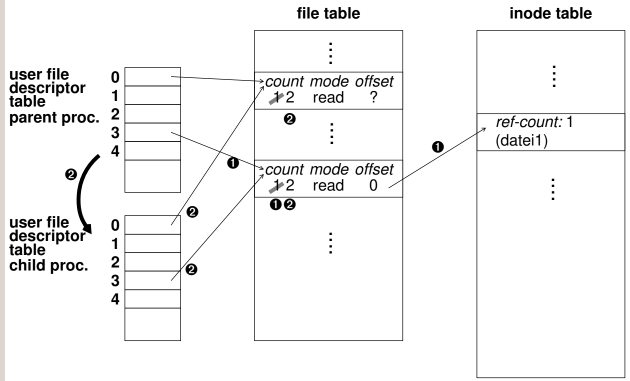
3 Kontrollstrukturen und Systemaufruf *dup( )*



- Umleiten von stdin:

```
❶ fd1 = open("datei", O_RDONLY);
❷ close(0);
❸ dup(fd1);
❹ close(fd1);
```


4 Kontrollstrukturen und Systemaufruf *fork( )*




```
❶ fd1 = open("datei", O_RDONLY);
❷ fork( );
```


J.4 Vnodes

1 Motivation

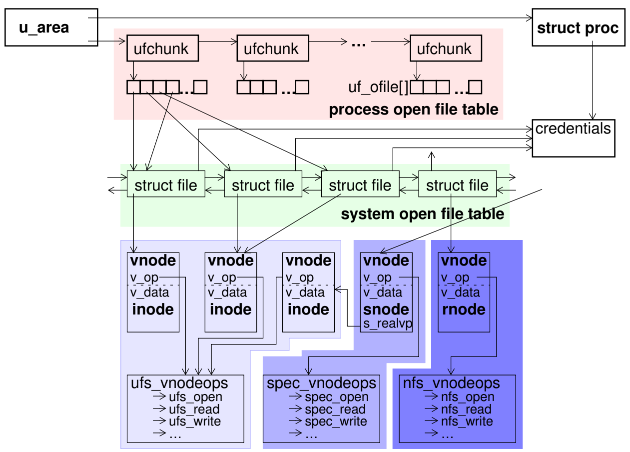
- Inode = konkrete Implementierung eines Dateiverwaltungsblocks eines speziellen Dateisystems
- Problem: unterschiedliche Dateisysteme in einem UNIX-System
  - ◆ S5 – SystemV-File-System
  - ◆ UFS - UNIX-File-System
  - ◆ NFS - Network File-System
  - ◆ specFS - special File-System (Gerätedateien, FIFO-Dateien)

➥ unterschiedliche Implementierung (Inode, *lookup*-Funktion, …)

1 Motivation (2)

- Lösung: **Vnode** = abstrakte Darstellung einer Datei
  - ➥ definiert Operationen auf Dateien
    - ➤ *open, close, read/write, lookup, …*
  - ➥ + allgemeine Zustandsinformationen
    - ➤ Dateityp (*regular file/directory*)
    - ➤ Referenzzähler
    - ➤ Verwaltungsinformationen für *locking* etc.
    - ➤ Zeiger auf konkrete Operationen und konkreten Zustand
- konkrete Darstellung einer Datei
  - ➥ Implementierung der Operationen
    - ➤ *ufs_open, ufs_close, ufs_read, ufs_lookup, …*
  - ➥ + konkreter Zustand
    - ➤ *in-core Inode*

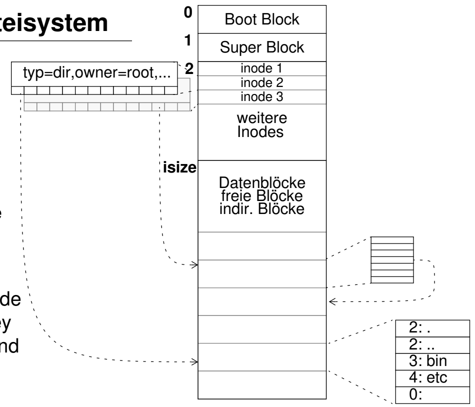
2 Kontrollstrukturen (SystemV.4)



J.5 Dateisysteme

Aufbau eines Dateisystems auf einer Disk

1 SystemV-Dateisystem



- Blockgröße 512 oder 1024 Byte

- große Unterschiede zwischen Berkeley fast-file-system und UNIX SystemV-Dateisystemen!

1 SystemV-Dateisystem (2)

- Super-Block
  - ◆ Der *Super-Block* enthält die Verwaltungsinformationen für ein Dateisystem
    - ➤ Größe
    - ➤ Anzahl der freien Blöcke
    - ➤ Liste freier Blöcke *(free block list)*
    - ➤ Zeiger auf nächsten freien Block in der *free block list*
    - ➤ Größe der Inode-Liste
    - ➤ Anzahl freier Inodes
    - ➤ Liste freier Inodes *(free inode list)*
    - ➤ Zeiger auf ersten freien Inode
    - ➤ Lock-Felder für *free block* und *free inode list*
    - ➤ *modified flag* für Super-Block

2 Berkeley Fast File System (UNIX File System — UFS)

- für UNIX 4.2bsd entwickelt - effizienter als SystemV-Dateisysteme
- Blockgröße minimal 4k, meist 8k
  - ⇒ über zweifach indirekte Blöcke können bis zu 64 GB adressiert werden
- Directories anders aufgebaut - Dateinamen bis zu 255 Zeichen lang
- Dateisystem in ***cylinder groups*** aufgeteilt (*cylinder group* = Gruppe von aufeinanderfolgenden Zylindern auf Platte, häufig 16)
- pro *cylinder group* werden folgende Daten gehalten:
  - ◆ Backup-Kopie des Superblocks
  - ◆ Bitmap für freie Blöcke (statt *free list*)
  - ◆ Inodes
  - ◆ Datenblöcke

2 Berkeley Fast File System (2)

- Um Verschnitt gering zu halten, können Blöcke in 2, 4 oder 8 ***Fragmente*** aufgeteilt werden, die einzeln adressierbar sind
- Eine Datei wird - wenn möglich - in einer *cylinder group* gehalten

Boot block
Super block
Cylinder group block
Inode list
Data blocks
Super block
Cylinder group block
Inode list
Data blocks

J.6 Virtuelles Dateisystem

1 Überblick

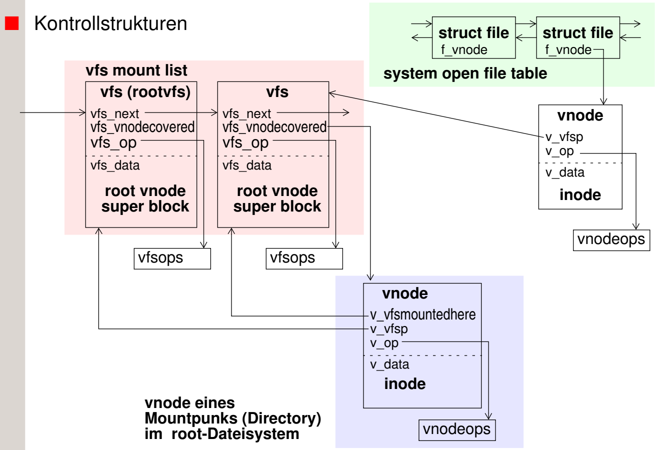
- Vnodes bilden Schnittstelle zu dateisystemunabhängigen Operationen auf Dateien
- das *Virtual File System* (**VFS**) stellt eine vergleichbare Schnittstelle für Operationen auf Dateisystemen zur Verfügung — z. B.
  - ◆ ***vfs_mount***, ***vfs_unmount***
  - ◆ ***vfs_statvfs*** Statistikdaten ermitteln
  - ◆ ***vfs_root*** root-vnode ermitteln
  - ◆ ***vfs_sync*** gepufferte Daten schreiben
- Implementierung durch VFS-switch-Tabelle
  - ◆ Dateisystemname, Initialisierungsfunktion, ***vfsops*** — Feld mit Zeigern auf die FS-Funktionen

2 Mount-Mechanismus

- über die mount-Funktion wird der root-Vnode eines Dateisystems auf einen Directory-Vnode des bestehenden Dateibaums montiert
  - ◆ Systemkern verwaltet die montierten Dateisysteme in verketteten *vfs*-Strukturen
  - ◆ über die vfs-Struktur kann ermittelt werden:
    - ➤ auf welchen Vnode das Dateisystem montiert ist
    - ➤ der root-Vnode des Dateisystems
  - ◆ Vnode des Mount-Punkts (Directory) enthält Eintrag mit Zeiger auf vfs-Datenstruktur des montierten Dateisystems

2 Mount-Mechanismus (2)

- Kontrollstrukturen

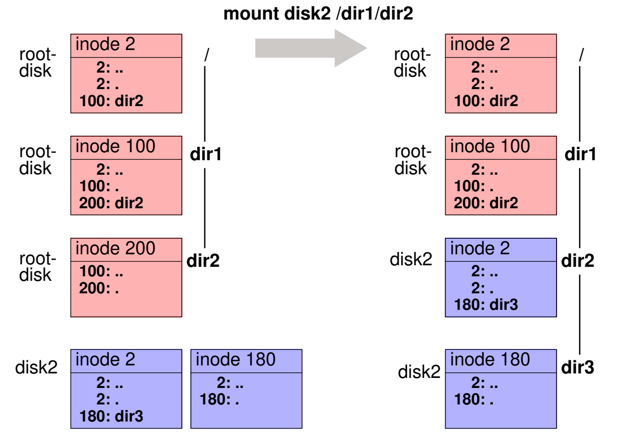


2 Mount-Mechanismus (3)

- *lookup*-Funktion des UNIX-Kerns berücksichtigt bei Pfad-Vnode-Umwandlung Mount-Punkte
  - ◆ ist in einem vnode *vfs_mountedhere* gesetzt, wird der Vnode beim Abstieg im Pfad durch den *root-vnode* des *VFS* ersetzt
  - ◆ ein *root-vnode* eines *VFS* wird bei Verfolgung von ".." durch *vfs_vnodecovered* des *VFS* ersetzt
  - ◆ an Directory-Einträgen ändert sich nichts

  ➥ an Mount-Punkten Diskrepanz zwischen Directory-Einträgen und Ergebnis von ***stat(2)***

2 Mount-Mechanismus (4)



J.7 Network File System (NFS)

1 Überblick

- 1984 von Sun Microsystems Inc. entwickelt
- NFS ermöglicht, Teilbäume von Dateisystemen anderer Rechner in das lokale Dateisystem einzuhängen *(**mounten**)*
- ein Teilbaum eines *Remote*-Dateisystems verhält sich dadurch genau so, wie eine Partition einer lokalen Platte, die mit einem ***mount***-Befehl in den Dateibaum eingehängt wurde
- die Dateien des *Remote*-Dateisystems können mit den gleichen Kommandos und Systemaufrufen wie lokale Dateien bearbeitet werden