2 Dateien und Kataloge

2.1 Was ist eine Datei? - Erläuterung der Betrachtungsweise

❑ Häufig zu findende Definition

- Für eine bestimmte Aufgabe bzw. unter bestimmten Gesichtspunkten in einem externen Speicher oder auf einem externen Datenträger zusammengefaßte Daten.

❑ Betriebssystembezogene Definition

- Benannte Ansammlung von Daten zusammen mit Operationen (Methoden), auf denen jede Manipulation der Ansammlung oder ihrer Komponenten aufbauen muß.

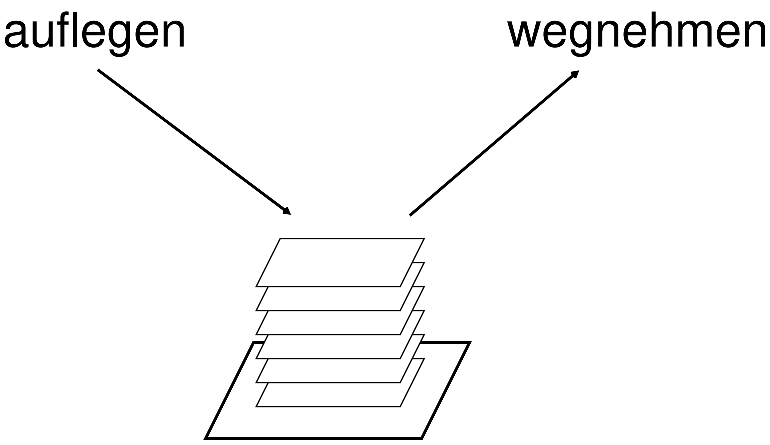
  D. h. einerseits ist eine Datei gegeben durch ihre **Daten und** die anwendbaren **Operationen**, andererseits ist eine Implementation der Operationen nur einmal im Betriebssystem vorhanden und nicht einmal pro Datei.

❑ Wie kann man dem in der strukturellen Erfassung von Betriebssystemen Rechnung tragen?

2.2 Systemtheoretischer Standpunkt

- ❏ System
  - Gesamtheit, Gefüge von Teilen, die voneinander abhängig sind, ineinandergreifen oder zusammenwirken.
  - Sollen die Teile ihrerseits als zusammengesetzt betrachtet werden, nennt man sie Subsysteme. Soll darüber keine Aussage gemacht werden, spricht man von Komponenten.
- ❏ Systembeschreibung und System
  - Bauplan - reales Exemplar
  - Beschreibung - Ausprägung
  - Begriff "Datei" - (existierende) Datei
  - Klasse - Instanz
- ❏ Systembeschreibungen bestehen aus der Angabe
  - der Menge Z möglicher Zustände
  - der Menge E möglicher Einflüsse aus der Systemumgebung
  - der Menge A möglicher Einflüssen auf die Systemumgebung
  - Zustandsübergangsfunktionen F_t:Z x E $\rightarrow$ Z x A
- ❏ Systemausprägungen
  - besitzen zu jedem Zeitpunkt einen Zustand
  - ändern ihren Zustand mit der Zeit gemäß der Zustandsübergangsfunktion und den Umgebungseinflüssen

❑ Beispiel: Der "Stapel"



Stapel $s \in S$

Teil $t \in T$

Funktionalität:

neuer_Stapel : $\rightarrow S$

auflegen : $S \times T \rightarrow S$

wegnehmen : $S \rightarrow S \times T$

Axiome:

wegnehmen(auflegen(s, t)) = <s, t>

......

Beispiel:

wegnehmen(

π_1(wegnehmen(auflegen(auflegen(s, t_1), t_2))))

= wegnehmen(auflegen(s, t_1))

= $\langle s, t_1 \rangle$

π_1 ist dabei die Projektion eines kartesischen Produktes auf seine erste Komponente

```
template <class element_type>
class stack
  {  element_type content[500];
     int          stackpointer;
  public:
     stack()
        {  stackpointer = 0;  };
     void push(element_type e)
        {  content[stackpointer++] = e;
           if (stackpointer >= 500)
              {  cerr << "\nStapel zu hoch\n";
                 exit(0);
              }
        };
     element_type pop()
        {  if (stackpointer == 0)
              {  cout << "\nStapel leer\n";
                 exit(0);
              }
           return content[--stackpointer];
        };
  };
```

Stapel int_stack von ganzen Zahlen:

```
stack<int> int_stack;
```

Stapel char_stack von Zeichen:

```
stack<char> char_stack;
```

Beispiel Saalbeleuchtung

Z = {ein, aus, Knopf_gedrückt, Knopf_nicht_gedrückt}

E = {Knopf_belasten, Knopf_entlasten}

A = {hell, dunkel}

ein,	Knopf_nicht_gedrückt;	Knopf_entlasten	$\rightarrow$	ein,	Knopf_nicht_gedrückt;	hell
ein,	Knopf_nicht_gedrückt;	Knopf_belasten	$\rightarrow$	aus,	Knopf_gedrückt;	dunkel
aus,	Knopf_gedrückt;	Knopf_belasten	$\rightarrow$	aus,	Knopf_gedrückt;	dunkel
aus,	Knopf_gedrückt;	Knopf_entlasten	$\rightarrow$	aus,	Knopf_nicht_gedrückt;	dunkel
aus,	Knopf_nicht_gedrückt;	Knopf_entlasten	$\rightarrow$	aus,	Knopf_nicht_gedrückt;	dunkel
aus,	Knopf_nicht_gedrückt;	Knopf_belasten	$\rightarrow$	ein,	Knopf_gedrückt;	hell
ein,	Knopf_gedrückt;	Knopf_belasten	$\rightarrow$	ein,	Knopf_gedrückt;	hell
ein,	Knopf_gedrückt;	Knopf_entlasten	$\rightarrow$	ein,	Knopf_nicht_gedrückt;	hell

```
enum Beleuchtungszustand {HELL, DUNKEL};

class Beleuchtung
  { enum {EIN, AUS}                                   Schaltzustand;
    enum {Knopf_gedrueckt, Knopf_nicht_gedrueckt} Schalterzustand;
  public:
    Beleuchtung()
      { Schaltzustand = EIN;
        Schalterzustand = Knopf_nicht_gedrueckt;
      }
    Beleuchtungszustand Knopf_entlasten()
      { Schalterzustand = Knopf_nicht_gedrueckt;
        return (Schaltzustand == EIN ? HELL : DUNKEL);
      };
    Beleuchtungszustand Knopf_belasten()
      { if (Schalterzustand == Knopf_nicht_gedrueckt) {
          Schalterzustand = Knopf_gedrueckt;
          Schaltzustand = (Schaltzustand == EIN ? AUS : EIN);
        }
        return (Schaltzustand == EIN ? HELL : DUNKEL);
      };
  };
```

```
#include "prolog.h"
```

. . . . . *Klasse Beleuchtung*

```
void main()
{ Beleuchtung  Tafel, Saal;
  char         operation;

  do {
    cout << "\noperation (<s|t>b = <Saal|Tafel>Schalter belasten"
         << "\n           <s|t>e = <Saal|Tafel>Schalter entlasten"
         << "\n           x      = exit)                           ";
    cin >> operation;
```

. . . . . *Reaktion*

```
  } while (1);
}
```

```
switch (operation) {
  case 's': { cin >> operation;
              switch (operation) {
                case 'b': cout << "\n===== Im Saal ist es "
                               << (Saal.Knopf_belasten()
                                     == HELL ? "hell." : "dunkel.")
                               << "\n" << flush;
                          break;
                case 'e': cout << "\n===== Im Saal ist es "
                               << (Saal.Knopf_entlasten()
                                      == HELL ? "hell." : "dunkel.")
                               << "\n" << flush;
                          break;
                default:  cout << "\n\n!!!!! Bedienungsfehler,"
                               << " falsche Operationsangabe. !!!!!\n"
                               << flush;
                          continue;
              }
              break;
            }
```

```
    case 't': { cin >> operation;
                switch (operation) {
                  case 'b': cout << "\n===== An der Tafel ist es "
                                 << (Tafel.Knopf_belasten()
                                       == HELL ? "hell." : "dunkel.")
                                 << "\n" << flush;
                            break;
                  case 'e': cout << "\n===== An der Tafel ist es "
                                 << (Tafel.Knopf_entlasten()
                                       == HELL ? "hell." : "dunkel.")
                                 << "\n" << flush;
                            break;
                  default:  cout << "\n\n!!!!! Bedienungsfehler,"
                                 << " falsche Operationsangabe. !!!!!\n"
                                 << flush;
                            continue;
                }
                break;
              }
    case 'x': exit(0);
    default:  cout << "\n\n!!!!! Bedienungsfehler,"
                   << " falsche Operationsangabe. !!!!!\n" << flush;
              continue;
  }
```

```
faui45a% beleuchtung

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              sb

===== Im Saal ist es dunkel.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              tb

===== An der Tafel ist es dunkel.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              te

===== An der Tafel ist es dunkel.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              tb

===== An der Tafel ist es hell.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              te

===== An der Tafel ist es hell.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              se

===== Im Saal ist es dunkel.

operation (<s|t>b = <Saal|Tafel>Schalter belasten
           <s|t>e = <Saal|Tafel>Schalter entlasten
           x      = exit)                              x
faui45a%
```

2.3 Dateien

❑ Aufgabe

- Speicherung großer Datenmengen
- Persistente Speicherung
- Nutzbarkeit für mehrere Benutzer

❑ Struktur

- Sequenz von Elementen eines Wertevorrates
- Partielle Funktion

```
enum structure_type {SEQUENTIAL, PARTIAL};

template <class          domain_type,
          class          range_type,
          structure_type structure>
  class function_type {
      ...
  public:
    int determine(domain_type argument,
                  range_type  &value);

    int define(domain_type argument,
               range_type  value);

    int get_length();
  };
```

❑ Zugriff

- sequentiell
- direkt

```
template<class key_type, class record_type, structure_type structure>
  class file_type
    { function_type</* with domain_type    */  key_type,
                    /* and range_ttype     */  record_type,
                    /* and structure_type */  structure>
      content;
    public:
      file_type(function_type<key_type, record_type, structure>
                fp_function)
        { content = fp_function;
        };
      int read(key_type key, record_type &value)
        { return content.determine(key, value);
        };
      int write(key_type key, record_type record)
        { return content.define(key, record))
        };
```

- ❏ Attribute
  - Definitionsbereich (z. B. int)
  - Wertebereich (z. B. char)
  - Zeitpunkt des Anlegens
  - Zeitpunkt des letzten Zugriffs
  - Zeitpunkt der letzten Modifikation
- ❏ Operationen
  - instantiieren (construct)
  - tilgen (destruct)
  - lesen (read)
  - schreiben (write)
  - anfügen (append)
  - Attribute lesen
  - Attribute modifizieren
  - ...

```
template<class key_type, class record_type, structure_type structure>
  class file_type
    {   function_type<key_type, record_type, structure> content;
      file_attributes_type attributes;
    public:
     file_type(function_type<key_type, record_type, structure> fp_function) {content = fp_function;};
      int read(key_type key, record_type &value) {return content.determine(key, value);};
      int write(key_type key, record_type record)
        { int result;
          if ((result = content.define(key, record)) >= 0)
            attributes.set_length(content.get_length());
          return result;
        };
      file_attributes_type *get_attributes()
        { file_attributes_type *result;
          result = new file_attributes_type;
          *result = attributes;
          return result;
        };
    };
```

❏ Speicherung

- zusammenhängend
- verkettet
- indiziert

❏ Übertragungsart

- synchron - asynchron (bezogen auf die virtuelle Zeit!)
- gepuffert - ungepuffert
- auf Anforderung - vorausschauend

2.4 Kataloge (Namensdienste, name server)

- ❏ Aufgabe
  - Vermittlung zwischen externer und interner Bezeichnung
- ❏ Namenssysteme
  - Literatur:

    Yeo, A. K.; Ananda, A. L.: A Taxonomy of Issues in Name Systems Design and Implementation. ACM Operating Systems Review, Vol. 27, No. 3, July 1993, pp. 4-18.

- Namen-systeme
  - Namen-dienste
    - Weiße Seiten
    - Gelbe Seiten
  - Namen-zuordnung
    - Namens-konventionen
      - unstrukturiert
      - strukturiert
        - attributiert
        - nicht-attributiert
    - Namens-raum
      - flach
      - Baum
      - gerichteter Graph
        - azyklisch
        - zyklisch
  - Namen-verteilung
    - zentral
    - verteilt
  - Mehrfache Namen-abbildung
    - Zweitname
    - Freundschaftsname (lokal eindeutig)
    - Kurzname
    - partieller Name
    - Mehrfachabstammung
    - Gruppenname (für 'multicasting')
    - Gattungsname (für 'load sharing')
  - Kontext
    - implizit
    - explizit
  - Namen-auflösung
    - Methode
      - absolut
      - inkrementell
    - Aktivität
      - Suchen
        - Dienst-struktur
          - statisch
          - Rundspruch
          - hybrid
        - Auflösung
          - rekursiv
          - iterativ
      - Interpretation
        - absolut
        - relativ
        - attributiert
          - geordnet
          - ungeordn.
      - Caching
        - lokal
        - regional
  - Datenbasen
    - Attribute
    - Aufbau
    - Anfragebeantwortung
  - Fehlertoleranz
  - Kommunikationsmechanismen
  - Zugriffskontrolle und Authentisierung

2.5 UNIX Dateisystem

Überblick — Benutzersicht

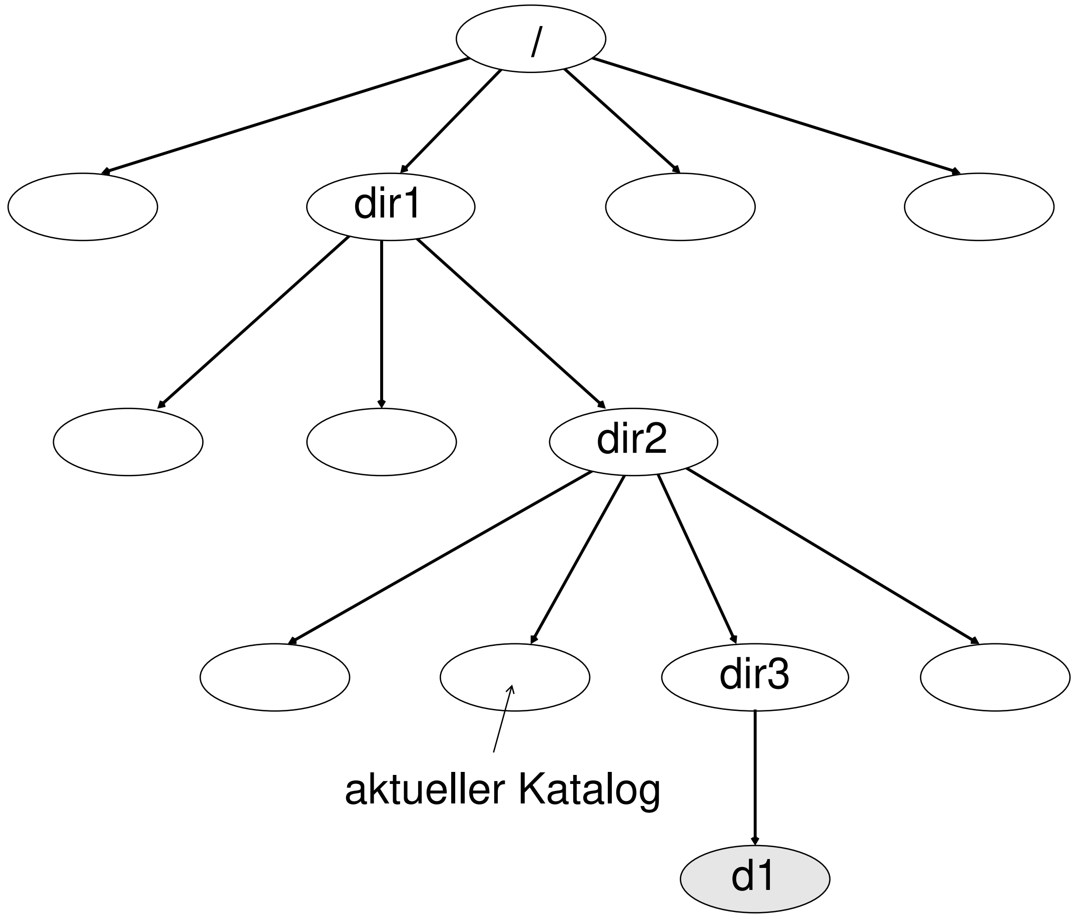
❑ Datei

- einfache, unstrukturierte Bytefolge
- kann beliebige Text- oder Binärinformation enthalten
- kann dynamisch erweitert werden

❑ Katalog

- baumförmig strukturiert
- Knoten sind **Dateikataloge** ***(Directories)***, ”Blätter” sind **Dateien** ***(Files)*** oder Verweise auf Dateien ***(Links)***
- der oberste Dateikatalog ist die **Wurzel** ***(Root)*** des Dateibaums
- jedem Benutzer ist zu jeder Zeit eine **aktuelle Wurzel** ***(current root)*** und ein **aktueller Dateikatalog** ***(current working directory)*** zugeordnet
- Peripheriegeräte (Terminals, Magnetbänder, ...) ***(special files)*** werden analog zu normalen Dateien ***(regular files)*** behandelt
- Dateien und Kataloge sind mit Zugriffsrechten versehen

❏ Pfade im Dateibaum



/dir1/dir2/dir3/d1 und **../dir3/d1** sind Pfade zu der Datei **d1**

- Wege durch den Dateibaum zu einem ”Blatt” werden durch **Pfade** beschrieben, Trennsymbol ist /
- Pfade = Aneinanderreihung der einzelnen Katalog-Namen —
  beginnend an der Wurzel **(** / **)** oder in dem aktuellen Katalog
- jeder Katalog enthält Verweise auf sich selbst **(.)** und auf den darüberliegenden Katalog **(..)** — in Pfadnamen verwendbar!

Funktionalität

❑ Abstraktionen für Benutzersicht:

1. Pfade, Dateinamen
2. Dateibaum, der sich über mehrere Platten (bzw. Partitionen) erstrecken kann
3. Dateien = unstrukturierte Bytefolgen
4. *special files* fast wie reguläre Dateien zugreifbar

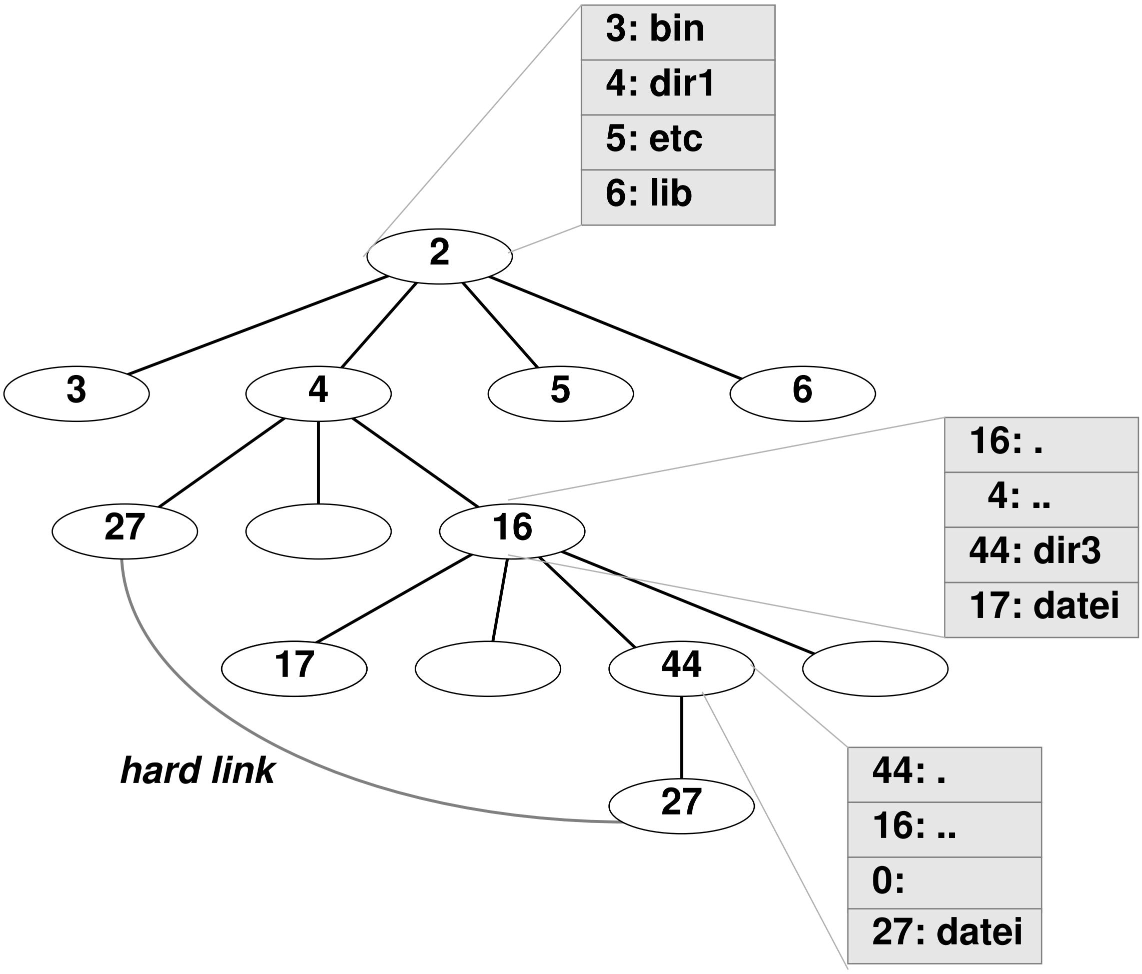
❑ diese Abstraktionen müssen vom Dateisystem auf konkrete Objekte abgebildet werden

❑ den obigen Abstraktionen entsprechen etwa folgende Objekte der Hardware bzw. des Betriebsystems:

1. Plattenpositionen, Adressen von Gerätecontrollern, etc.
2. wie 1.
3. Sammlung von Speicherblöcken auf externen Speichern (z. B. Plattenspeichern)
4. unterschiedliche Gerätetreiber für E/A-Operationen auf verschiedenen Peripheriegeräten

Überblick — Realisierung

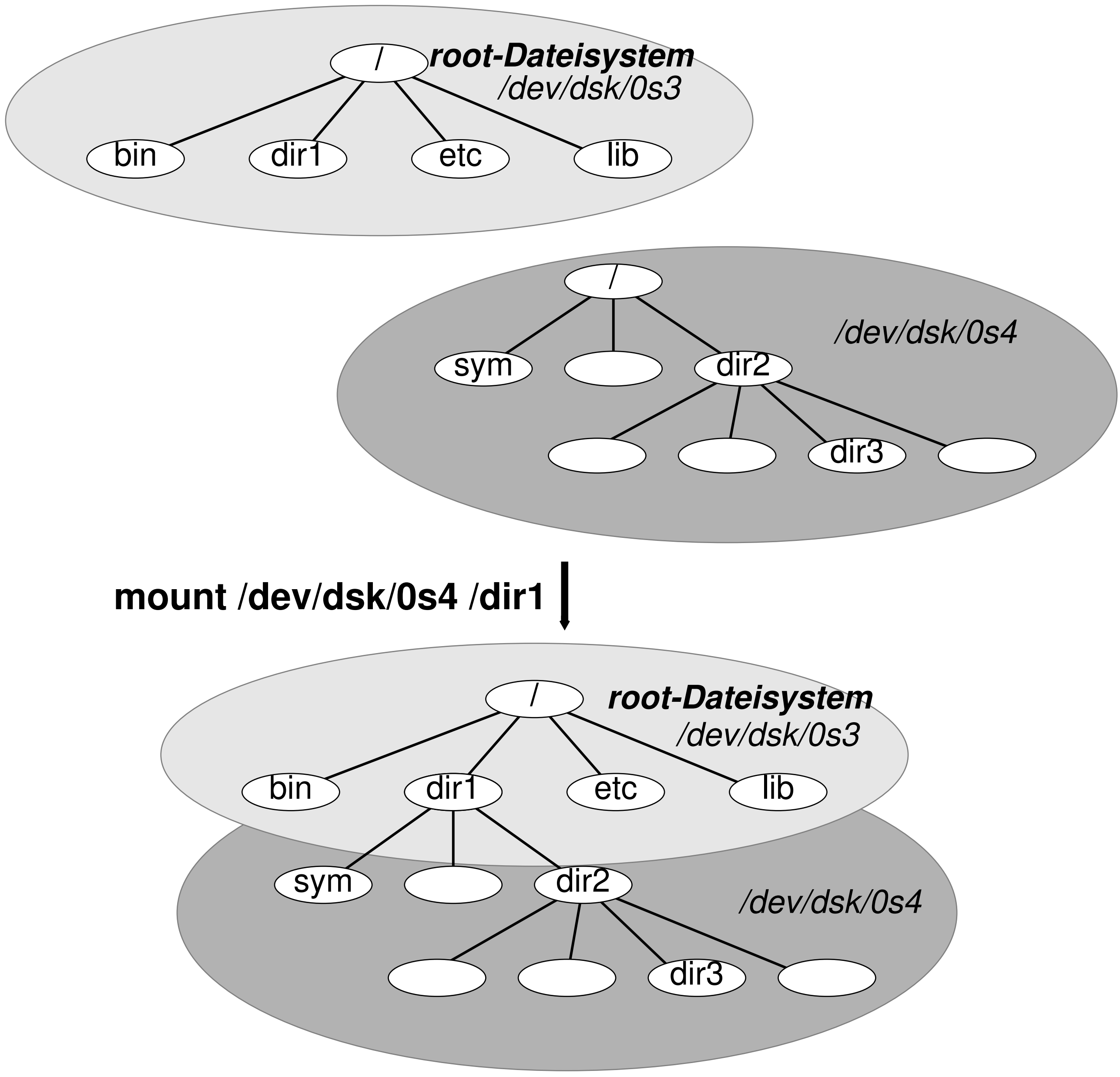
- Attribute einer Datei und Ortsinformationen über ihren Inhalt werden in einer Verwaltungsstruktur (***inode***) gehalten
- Flacher Namensraum für Inodes: durchnumeriert
- **Directories** ermöglichen Zuordnung ***Dateiname*** ⇒ ***inode***-Nr.
- es können mehrere Dateinamen auf den gleichen Inode verweisen (nur innerhalb eines Dateisystems!): ***(hard) links***



❏ der UNIX-Dateibaum kann aus mehreren Dateisystemen - auch auf verschiedenen physikalischen Platten - bestehen

❏ jedes Dateisystem besteht aus einem Dateibaum mit einer ***root***

❏ durch ***mounten*** kann die Root eines Dateibaums an einen beliebigen Knoten eines anderen (bereits gemounteten) Dateibaums angebunden werden



Operationen auf Directories

- ❏ lesen

  get directory entries in a filesystem independent format

  ```
  int getdents(int fd, char *buf, int nbytes)
  ```

  (oder getdirentries)

  - der Aufbau der Directory-Struktur kann aus `<dir.h>`, `<sys/dirent.h>` bzw. `<sys/dir.h>` übernommen werden
  - zum einfacheren Umgang mit Directories gibt es in der Regel Bibliotheksfunktionen: *opendir( ), readdir( ), ...,* siehe ***directory(3)***

- ❏ schreiben

  generell nicht gestattet, Einträge erfolgen nur durch Betriebssystem

- ❏ anlegen

  make a directory file

  ```
  int mkdir(char *path, int mode)
  ```

- ❏ löschen

  remove a directory file

  ```
  int rmdir(char *path)
  ```

- ändern von Einträgen

name of a file

```
int rename(char *path1, char *path2)
```

make a hard link to a file

```
int link(char *path1, char *path2)
```

remove directory entry

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

- Datei binden bzw. Bindung lösen

open or create a file for reading or writing

```
int open(char *path, int flags
        [ , int mode ] )
```

delete a descriptor

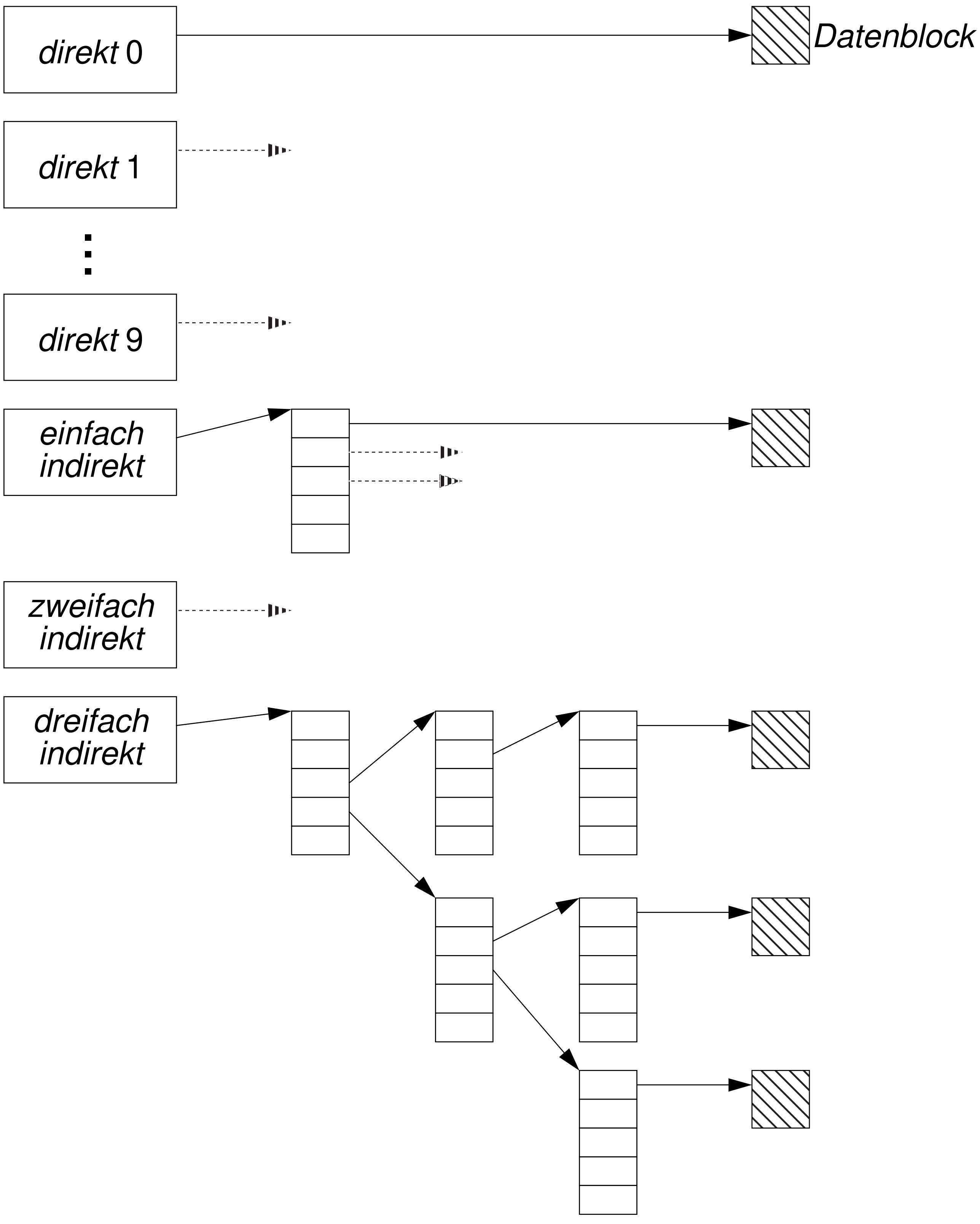
```
int close(int fd)
```

Inodes (Dateideskriptoren)

- ❑ ein Inode enthält alle Informationen, die zur Verwaltung einer Datei und zum Auffinden ihres Inhalts benötigt werden
- ❑ ein Inode enthält folgende Einträge
    - Dateityp
    - Eigentümer und Gruppenzugehörigkeit
    - Zugriffsrechte
    - Zugriffszeiten
        - letzte Änderung (*mtime*)
        - letzter Zugriff (*atime*)
        - letzte Änderung des Inodes (*ctime*)
    - Anzahl der (hard) Links auf die Datei
    - Dateigröße (in Bytes)
    - Adressen für Datenblöcke auf der Platte:
      (10 direkte Adressen + 3 Adressen für indirekte Adressierung)
- ❑ Datenstrukturen definiert in
  `/usr/include/ufs/inode.h` oder
  `/usr/include/sys/*inode.h` (systemabhängig!)

❑ Die Attributstruktur stat

```
struct    stat {
    dev_t       st_dev;         /* device file resides on */
    ino_t       st_ino;         /* the file serial number */
    mode_t      st_mode;        /* file mode */
                                0170000    /* type of file */
                                0040000    /* directory */
                                0020000    /* character special */
                                0060000    /* block special */
                                0100000    /* regular */
                                0120000    /* symbolic link */
                                0140000    /* socket */
                                0010000    /* fifo */
    nlink_t     st_nlink;       /* number of hard links to the file */
    uid_t       st_uid;         /* user ID of owner */
    gid_t       st_gid;         /* group ID of owner */
    dev_t       st_rdev;        /* the device identifier (special files
                                only)*/
    off_t       st_size;        /* total size of file, in bytes */
    time_t      st_atime;       /* file last access time */
    int         st_spare1;
    time_t      st_mtime;        /* file last modify time */
    int         st_spare2;
    time_t      st_ctime;       /* file last status change time */
    int         st_spare3;
    long        st_blksize;     /* preferred blocksize for file system
                                I/O*/
    long        st_blocks;      /* actual number of blocks allocated
                                */
    long        st_spare4[2];
};
```



- ❏ Adressierung der Datenblöcke

Betriebssystemschnittstelle für Dateien

❑ Zugriff

- sequentiell

  read input

  ```
  int read(int fd, char *buf,  int nbyte)
  ```

  read input and scatter

  ```
  int readv(int fd,
            struct iovec *iov, int iovcnt)
  ```

  *iov: Liste von Paaren bestehend aus Basis-adresse und Längenangabe

  iovcnt: Zahl der Listenelemente

  write output

  ```
  int write(int fd, char *buf,  int nbyte)
  ```

  write output and gather

  ```
  int writev(int fd,
             struct iovec *iov, int iovcnt)
  ```

- direkt

  durch Voreinstellung der Adresse

  move read/write pointer

  ```
  off_t lseek(int fd,
              off_t offset, int whence)
  ```

❑ Attribute einstellen

- Länge

set a file to a specified length

```
int truncate(char *path, off_t length)
int ftruncate(int fd, off_t length)
```

- Zeitattribute

set access and modification times of a file

```
int utimes(char *file, struct timeval *tvp)
```

- Zugriffsattribute Z

$Z \in \{\text{Besitzer, Gruppe, alle}\}$
$\times \{\text{lesen, schreiben, ausführen}\}$

set file creation mode mask

```
int umask(int mask)
```

change owner and group of a file

```
int chown(char *path, int owner, int group)
int fchown(int fd, int owner, int group)
```

change mode of file

```
int chmod(char *path, mode_t mode)
int fchmod(int fd, mode_t mode)
```

❑ Attribute erfragen

get file status

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