2 Operationen auf Dateien (2)

- Positionieren des Schreib-/Lesezeigers *(Seek)*
  - Identifikation der Datei
  - In manchen 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 Katalog und Freigabe der Plattenblocks

C.2 Kataloge

- Ein Katalog gruppiert Dateien und evtl. andere Kataloge
  - Verknüpfung mit der Benennung
    - Katalog enthält Namen und Verweise auf Dateien und andere Kataloge
      z.B. *UNIX, MS-DOS*
  - Zusätzliche Bedingung
    - Katalog enthält Namen und Verweise auf Dateien, die einer bestimmten Zusatzbedingung gehorchen
      z.B. gleiche Gruppennummer in *CP/M*
      z.B. eigenschaftsorientierte und dynamische Gruppierung in *BeOS-BFS*
- Katalog erlaubt die Benennung von Dateien
  - Vermittlung zwischen externer und interner Bezeichnung (Dateiname — Plattenblocks)

1 Operationen auf Katalogen

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

2 Attribute von Katalogen

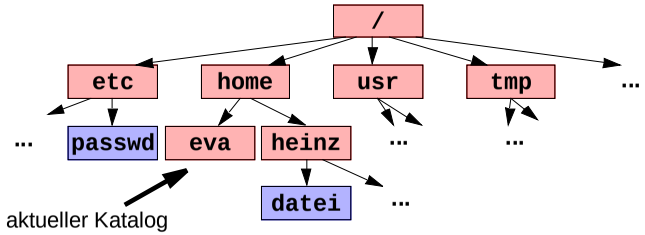
- Die meisten Dateiattribute treffen auch für Kataloge zu
  - Name, Ortsinformationen, Größe, Zeitstempel, Rechte, Eigentümer

C.3 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

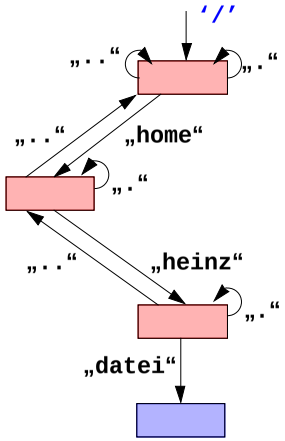
- Baumstruktur



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

1 Pfadnamen (2)

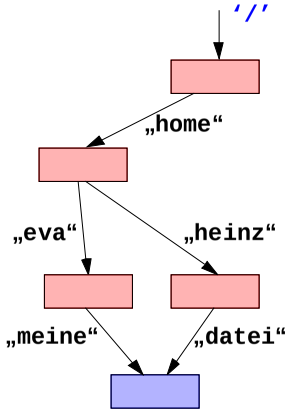
- Eigentliche Baumstruktur



- 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 („**..**“)

1 Pfadnamen (3)

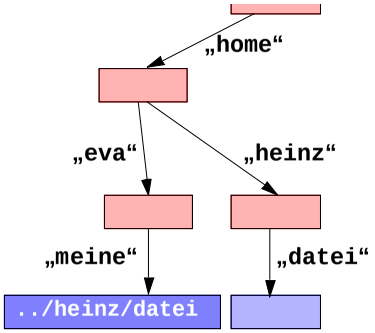
- Links *(Hard links)*
  - Dateien können mehrere auf sich zeigende Verweise besitzen, sogenannte Hard links (nicht jedoch Kataloge)



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

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.

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 (nur vom Eigentümer veränderbar)
  - 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
  - Schreibrecht ist einschränkbar auf eigene Dateien

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

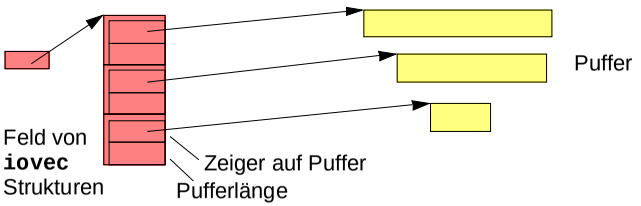
```
int close( int fd );
```

- Fehlermeldungen
  - Anzeige durch Rückgabe von **`-1`**
  - Variable **`errno`** enthält Fehlercode

3 Dateien (2)

- Weitere Operationen
  - Lesen und Schreiben in Pufferlisten

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



  - Positionieren des Schreib-, Lesezeigers

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


3 Dateien (3)

- Attribute einstellen
  - Länge

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

  - Zugriffs- und Modifikationszeiten

```
int utimes( char *path, struct timeval *tvp );
```

  - Implizite Maskierung von Rechten

```
mode_t umask( mode_t mask );
```

  - Eigentümer und Gruppenzugehörigkeit

```
int chown( char *path, uid_t owner, gid_t group );
int lchown( char *path, uid_t owner, gid_t group );
int fchown( int fd, uid_t owner, gid_t group );
```

3 Dateien (4)

- Zugriffsrechte

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

- Alle Attribute abfragen

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


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:

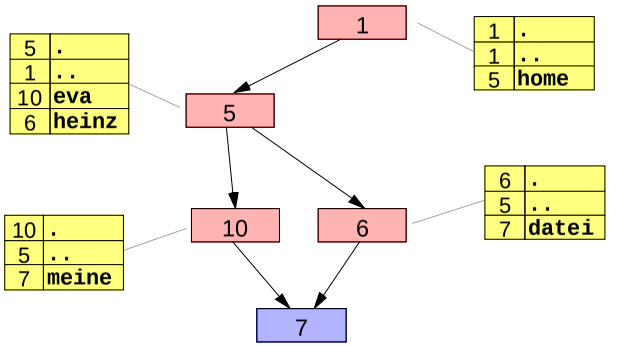
```
DIR *opendir( const char *path );
struct dirent *readdir( DIR *dirp );
int closedir( DIR *dirp );
```

- Symbolische Namen auslesen

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


5 Inodes

- Attribute einer Datei und Ortsinformationen über ihren Inhalt werden in sogenannten Inodes gehalten
  - Inodes werden pro Partition numeriert *(Inode number)*
- Kataloge enthalten lediglich Paare von Namen und Inode-Nummern

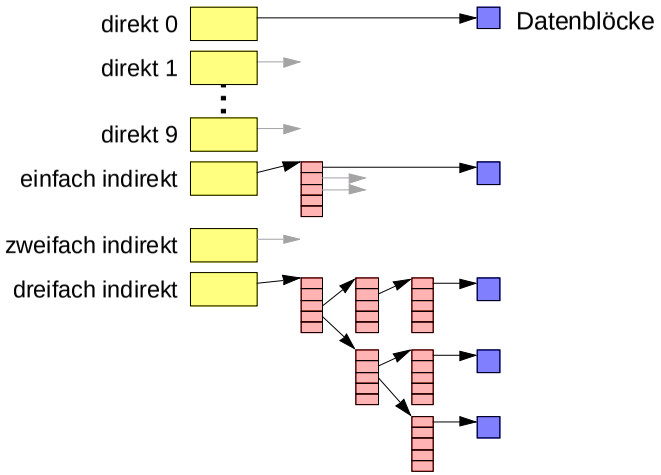


5 Inodes (2)

- Inhalt eines Inodes
  - Inodenummer
  - Dateityp: Katalog, 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)
  - Adressen der Datenblöcke des Datei- oder Kataloginhalts (zehn direkt Adressen und drei indirekte)

5 Inodes (3)

- Adressierung der Datenblöcke



6 Spezialdateien

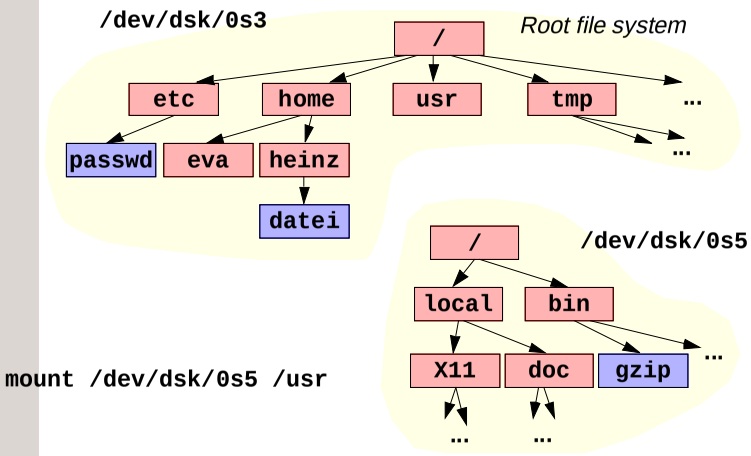
- Periphere Geräte werden als Spezialdateien repräsentiert
  - Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
  - Öffnen der Spezialdateien schafft eine (evtl. exklusive) Verbindung zum Gerät, die durch einen Treiber hergestellt wird
- Blockorientierte Spezialdateien
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

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)

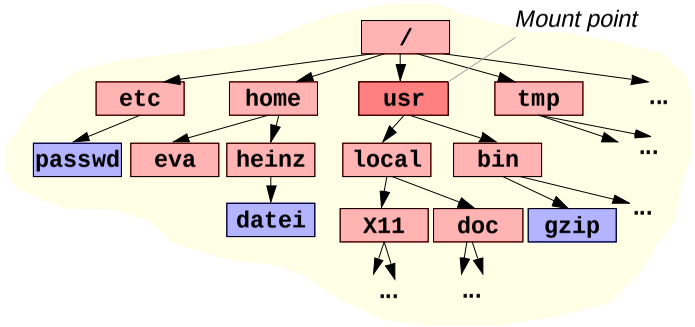
- Beispiel




```
mount /dev/dsk/0s5 /usr
```


7 Montieren des Dateibaums (2)

- Beispiel nach Ausführung des Montierbefehls



C.4 Beispiel: Windows 95 (PM/FAT)

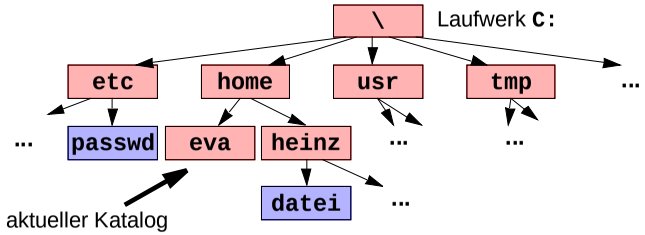
- PM/FAT = Protected mode/File allocation table
  - MS-DOS–kompatibles Dateisystem
- Datei
  - einfache, unstrukturierte Folge von Bytes
  - beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
  - dynamisch erweiterbar
  - Zugriffsrechte: lesbar, schreib- und lesebar

C.4 Beispiel: Windows 95 (PM/FAT) (2)

- Katalog
  - baumförmig strukturiert
    - Knoten des Baums sind Kataloge
    - Blätter des Baums sind Dateien
  - jedem Windows Programm ist zu jeder Zeit ein aktuelles Laufwerk und ein aktueller Katalog pro Laufwerk zugeordnet
  - Zugriffsrechte: lesbar, schreib- und lesbar
- Partitionen heißen Laufwerke
  - Sie werden durch einen Buchstaben dargestellt (z.B. `C:`)

1 Pfadnamen

- Baumstruktur



- Pfade
  - z.B. „`C:\home\heinz\datei`“, „`\tmp`“, „`C:..\heinz\datei`“
  - „\“ ist Trennsymbol *(Backslash)*; beginnender „\“ bezeichnet Wurzelkatalog; sonst Beginn implizit mit dem aktuellem Katalog
  - beginnt der Pfad ohne Laufwerksbuchstabe wird das aktuelle Laufwerk verwendet

1 Pfadnamen (2)

- Namenskonvention
  - Kompatibilitätsmodus: 8 Zeichen Name, 3 Zeichen Erweiterung (z.B. **`AUTOEXEC.BAT`**)
  - Sonst: 255 Zeichen inklusive Sonderzeichen (z.B. „**`Eigene Programme`**“)
- Kataloge
  - Jeder Katalog enthält einen Verweis auf sich selbst („`.`“) und einen Verweis auf den darüberliegenden Katalog im Baum („`..`“) (Ausnahme Wurzelkatalog)
  - keine Hard links oder symbolischen Namen

2 Rechte

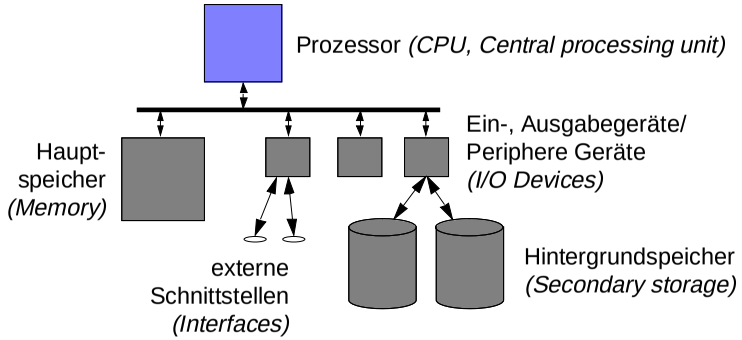
- Rechte pro Datei und Katalog
  - schreib- und lesbar — nur lesbar *(read only)*
- Keine Benutzeridentifikation
  - Rechte garantieren keinen Schutz, da veränderbar

3 Dateien

- Attribute
  - Name, Dateilänge
  - Attribute: versteckt *(hidden)*, archiviert *(Archive)*, Systemdatei *(System)*
  - Rechte
  - Ortsinformation: Nummer des ersten Plattenblocks
  - Zeitstempel: Erzeugung, letzter Schreib- und Lesezugriff

D Prozesse und Nebenläufigkeit

- Einordnung



D.1 Prozessor

- Register
  - Prozessor besitzt Steuer- und Vielzweckregister
  - Steuerregister:
    - Programmzähler *(Instruction pointer)*
    - Stapelregister *(Stack pointer)*
    - Statusregister
    - etc.
- Programmzähler enthält Speicherstelle der nächsten Instruktion
  - Instruktion wird geladen und
  - ausgeführt
  - Programmzähler wird inkrementiert
  - dieser Vorgang wird ständig wiederholt

D.1 Prozessor (2)

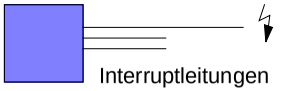
- Beispiel für Instruktionen

```
...
0010 5510000000  movl DS:$10, %ebx
0015 5614000000  movl DS:$14, %eax
001a 8a          addl %eax, %ebx
001b 5a18000000  movl %ebx, DS:$18
...
```

- Prozessor arbeitet in einem bestimmten Modus
  - Benutzermodus: eingeschränkter Befehlssatz
  - privilegierter Modus: erlaubt Ausführung privilegierter Befehle
    - Konfigurationsänderungen des Prozessors
    - Moduswechsel
    - spezielle Ein-, Ausgabebefehle

D.1 Prozessor (3)

- Unterbrechungen *(Interrupts)*



Signalisieren der Unterbrechung
(Interrupt request; IRQ)

  - Prozessor unterbricht laufende Bearbeitung und führt eine definierte Befehlsfolge aus (vom privilegierten Modus aus konfigurierbar)
  - vorher werden alle Register einschließlich Programmzähler gesichert (z.B. auf dem Stack)
  - nach einer Unterbrechung kann der ursprüngliche Zustand wiederhergestellt werden
  - Unterbrechungen werden im privilegierten Modus bearbeitet

D.1 Prozessor (4)

- Systemaufrufe *(Traps; User interrupts)*
  - Wie kommt man kontrolliert vom Benutzermodus in den privilegierten Modus?
  - spezielle Befehle zum Eintritt in den privilegierten Modus
  - Prozessor schaltet in privilegierten Modus und führt definierte Befehlsfolge aus (vom privilegierten Modus aus konfigurierbar)
  - solche Befehle werden dazu genutzt die Betriebssystemschnittstelle zu implementieren *(Supervisor calls)*
  - Parameter werden nach einer Konvention übergeben (z.B. auf dem Stack)

D.2 Prozesse

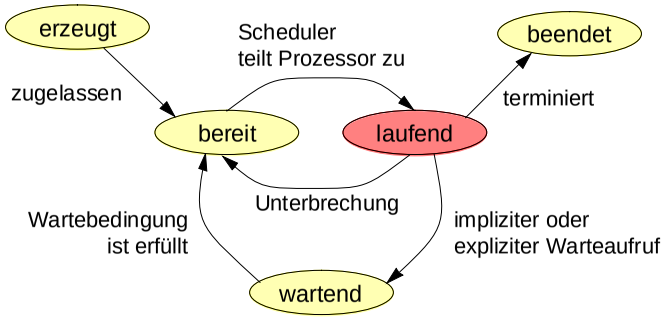
- Stapelsysteme *(Batch systems)*
  - ein Programm läuft auf dem Prozessor von Anfang bis Ende
- Heutige Systeme *(Time sharing systems)*
  - mehrere Programme laufen gleichzeitig
  - Prozessorzeit muß den Programmen zugeteilt werden
  - Programme laufen nebenläufig
- Terminologie
  - **Programm:** Folge von Anweisungen (hinterlegt beispielsweise als Datei auf dem Hintergrundspeicher)
  - **Prozeß:** Programm, das sich in Ausführung befindet, und seine Daten (ein Programm kann sich mehrfach in Ausführung befinden)

1 Prozeßzustände

- Ein Prozeß befindet sich üblicherweise in einem der folgenden Zustände:
  - **Erzeugt** *(New)*
    Prozeß wurde erzeugt; Prozeß besitzt noch nicht alle Betriebsmittel zum Laufen
  - **Bereit** *(Ready)*
    Prozeß besitzt alle nötigen Betriebsmittel und ist bereit zum Laufen
  - **Laufend** *(Running)*
    Prozeß wird vom realen Prozessor ausgeführt
  - **Blockiert/Wartend** *(Blocked/Waiting)*
    Prozeß wartet auf ein Ereignis (z.B. Fertigstellung einer Ein- oder Ausgabeoperation, Zuteilung eines Betriebsmittels, Empfang einer Nachricht)
  - **Beendet** *(Terminated)*
    Prozeß ist beendet; einige Betriebsmittel sind jedoch noch nicht freigegeben oder Prozeß muß aus anderen Gründen im System verbleiben

1 Prozeßzustände (2)

- Zustandsdiagramm

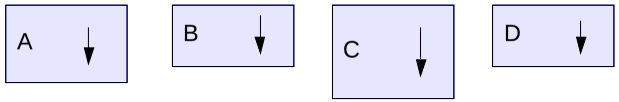


Nach Silberschatz, 1994

- Scheduler ist der Teil des Betriebssystems, der die Zuteilung des realen Prozessors vornimmt.

2 Prozeßwechsel

- Konzeptionelles Modell

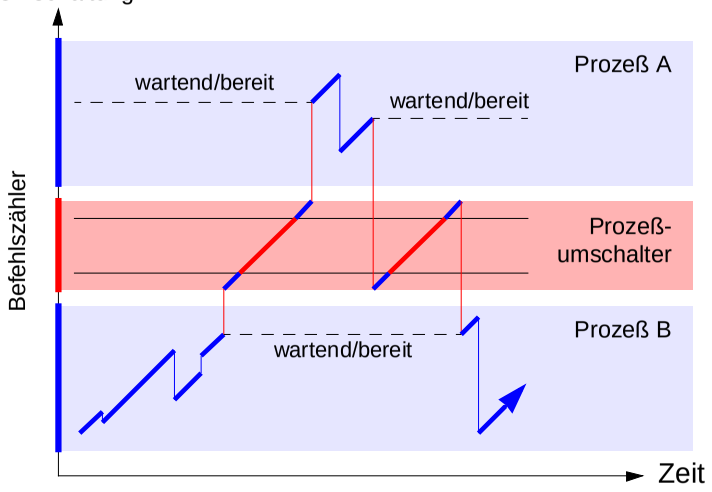


vier Prozesse mit eigenständigen Befehlszählern

- Umschaltung *(Context switch)*
  - Sichern der Register des laufenden Prozesses inkl. Programmzähler (Kontext),
  - Auswahl des neuen Prozesses,
  - Ablaufumgebung des neuen Prozesses herstellen (z.B. Speicherabbildung, etc.),
  - Gesicherte Register laden und
  - Prozessor aufsetzen.

2 Prozeßwechsel (2)

- Umschaltung



2 Prozeßwechsel (3)

- Prozeßkontrollblock *(Process control block; PCB)*
  - Datenstruktur, die alle nötigen Daten für einen Prozeß hält. Beispielsweise in UNIX:
    - Prozeßnummer *(PID)*
    - verbrauchte Rechenzeit
    - Erzeugungszeitpunkt
    - Kontext (Register etc.)
    - Speicherabbildung
    - Eigentümer *(UID, GID)*
    - Wurzelkatalog, aktueller Katalog
    - offene Dateien
    - ...

2 Prozeßwechsel (4)

- Prozeßwechsel unter Kontrolle des Betriebssystems
  - Mögliche Eingriffspunkte:
    - Systemaufrufe
    - Unterbrechungen
  - Wechsel nach/in Systemaufrufen
    - Warten auf Ereignisse (z.B. Zeitpunkt, Nachricht, Lesen eines Plattenblock)
    - Terminieren des Prozesses
  - Wechsel nach Unterbrechungen
    - Ablauf einer Zeitscheibe
    - bevorzugter Prozeß wurde laufbereit
- Auswahlstrategie zur Wahl des nächsten Prozesses
  - *Scheduler*-Komponente

3 Operationen auf Prozessen (Solaris)

- Typische Operationen eines UNIX Betriebssystems
  - Erzeugen eines neuen Prozesses (dupliziert den gerade laufenden Prozeß)

```
pid_t fork( void );
```

Vater

```
pid_t p;
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p != (pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
    ...
}
```

3 Operationen auf Prozessen (Solaris)

- Typische Operationen eines UNIX Betriebssystems
  - Erzeugen eines neuen Prozesses (dupliziert den gerade laufenden Prozeß)

```
pid_t fork( void );
```

Vater

```
pid_t p;
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p != (pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
    ...
}
```

Kind

```
pid_t p;
...
p= fork();
if( p == (pid_t)0 ) {
    /* child */
    ...
} else if( p != (pid_t)-1 ) {
    /* parent */
    ...
} else {
    /* error */
    ...
}
```


3 Operationen auf Prozessen (2)

- Ausführen eines Programms

```
int execve( const char *path, char *const argv[],
            char *const envp[] );
```

- Prozeß beenden

```
void _exit( int status );
[ void exit( int status ); ]
```

- Prozeßidentifikator

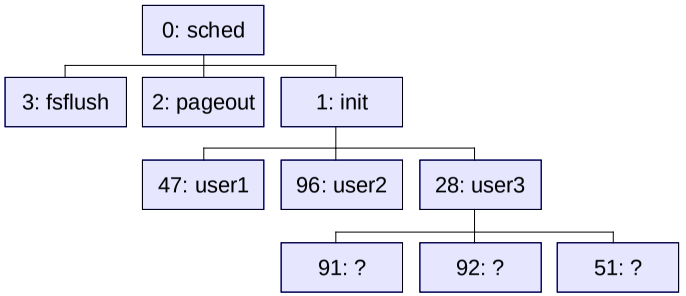
```
pid_t getpid( void );     /* eigene PID */
pid_t getppid( void );    /* PID des Vaterprozesses */
```

- Warten auf Beendigung eines Kindprozesses

```
pid_t wait( int *statusp );
```


4 Prozeßhierarchie (Solaris)

- Hierarchie wird durch Vater-Kind–Beziehung erzeugt



Frei nach Silberschatz 1994

  - Nur der Vater kann auf das Kind warten
  - Init-Prozeß adoptiert verwaiste Kinder

5 Aktivitätsträger *(Threads)*

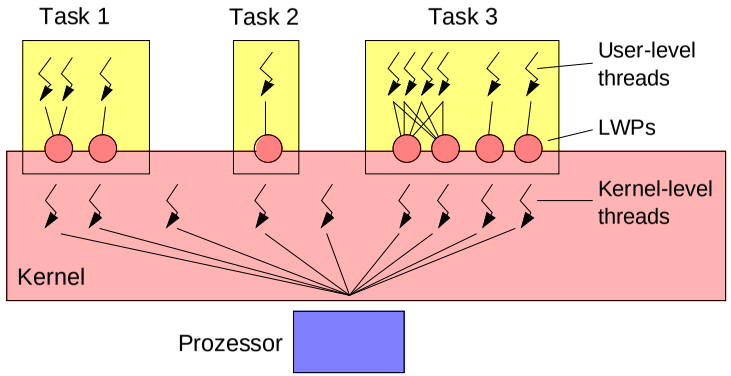
- Thread als leichtgewichtiger Prozeß *(Leight weight process, LWP)*
  - keine eigene Speicherabbildung
  - mehrere Threads teilen sich einen Speicherbereich und arbeiten im gleichen Instruktionen- und Datenbereich
  - Gruppe von Threads einschließlich ihrer Speicherabbildung wird oft als *Task* bezeichnet
- ★ Ein Prozeß ist ein Task mit einem Thread
- Implementierungen von Threads
  - User-level threads
  - Kernel-level threads

5 Aktivitätsträger (2)

- User-level threads
  - ◆ Instruktionen im Anwendungsprogramm schalten zwischen den Threads hin- und her (ähnlich wie der Scheduler im Betriebssystem)
  - ◆ Betriebssystem sieht nur einen Thread
    - keine Systemaufrufe zum Umschalten erforderlich
    - effiziente Umschaltung
  - ◆ Bei blockierenden Systemaufrufen bleiben alle User-level threads stehen
- Kernel-level threads
  - ◆ Betriebssystem schaltet Threads um
    - weniger effizientes Umschalten
  - ◆ Kein Blockieren unbeteiligter Threads bei blockierenden Systemaufrufen
  - ◆ Fairneßverhalten nötig (zwischen Prozeß und Task mit vielen Threads)

6 Beispiel: LWPs und Threads (Solaris)

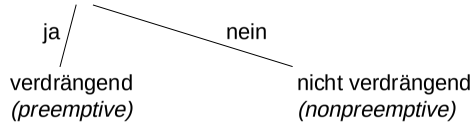
- Solaris kennt Kernel- und User-level threads



Nach Silberschatz, 1994

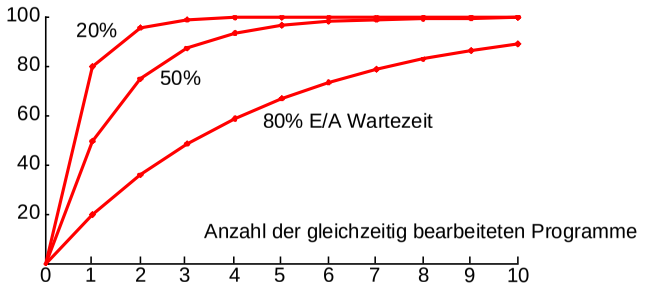
D.3 Auswahlstrategien

- Strategien zur Auswahl des nächsten Prozesses *(Scheduling strategies)*
  - ◆ Mögliche Stellen zum Treffen von Schedulingentscheidungen
    1. Prozeß wechselt vom Zustand „laufend“ zum Zustand „wartend“ (z.B. Ein-, Ausgabeoperation)
    2. Prozeß wechselt von „laufend“ nach „bereit“ (z.B. bei einer Unterbrechung des Prozessors)
    3. Prozeß wechselt von „wartend“ nach „bereit“
    4. Prozeß terminiert
  - ◆ Keine Wahl bei 1. und 4.
  - ◆ Wahl bei 2. und 3.



D.3 Auswahlstrategien (2)

- CPU Auslastung
  - ◆ CPU soll möglichst vollständig ausgelastet sein
- ★ CPU-Nutzung in Prozent, abhängig von der Anzahl der Programme und deren prozentualer Wartezeit



Nach Tanenbaum, 1995