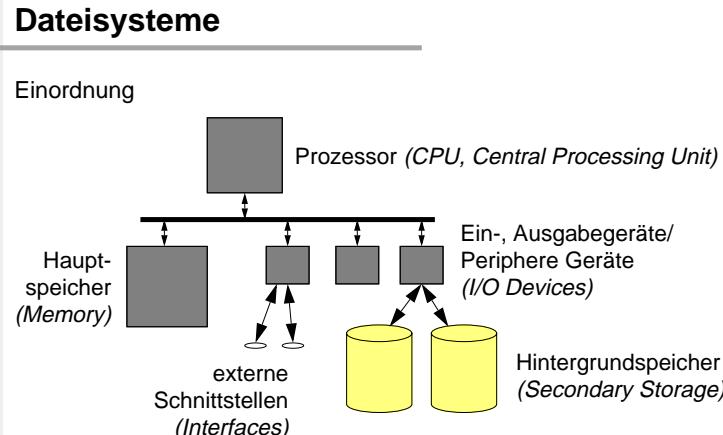


# C Dateisysteme

Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C - 1



Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C - 2

## C Dateisysteme (2)

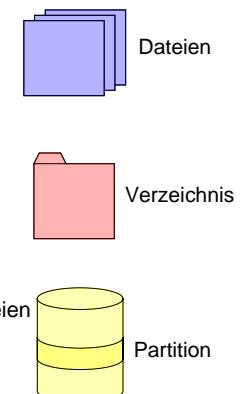
- Dateisysteme speichern Daten und Programme persistent in Dateien
  - ◆ Betriebssystemabstraktion zur Nutzung von Hintergrundspeichern (z.B. Platten, CD-ROM, Floppy Disk, Bandlaufwerke)
    - Benutzer muss sich nicht um die Ansteuerungen verschiedener Speichermedien kümmern
    - einheitliche Sicht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

Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C - 3

## C Dateisysteme (3)

- Datei
  - ◆ speichert Daten oder Programme
- Verzeichnis
  - ◆ fasst Dateien (u. Verzeichnisse) zusammen
  - ◆ erlaubt Benennung der Dateien
  - ◆ enthält Zusatzinformationen zu Dateien
- Partitionen
  - ◆ 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



Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C - 4

## 1 Dateien

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

### 1.1 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

### 1.1 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, Entwicklungswerzeuge, 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)

## 1.2 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.
  - ◆ eventuelle Anpassung der Attribute, z.B. Länge
- Lesen (*Read*)
  - ◆ Identifikation der Datei
  - ◆ Daten werden von Platte gelesen.

### 1.2 Operationen auf Dateien (2)

- Positionieren des Schreib-/Lesezeigers (*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 Katalog und Freigabe der Plattenblocks

## 2 Verzeichnisse / Kataloge

- Ein Verzeichnis gruppiert Dateien und evtl. andere Verzeichnisse
- Gruppierungsalternativen:
  - ◆ Verknüpfung mit der Benennung
    - Verzeichnis enthält Namen und Verweise auf Dateien und andere Verzeichnisse, z.B. *UNIX, MS-DOS*
  - ◆ 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 — Plattenblöcken)

## 2.1 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*)

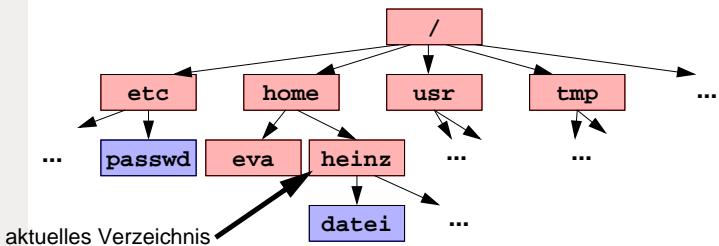
## 2.2 Attribute von Verzeichnissen

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

## 3 Beispiel: UNIX (Sun-UFS)

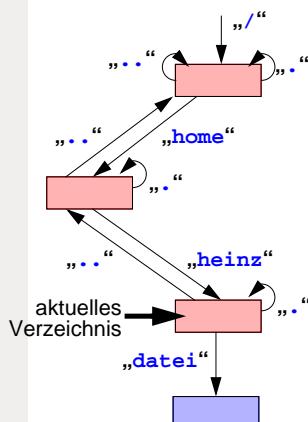
- Datei
  - ◆ einfache, unstrukturierte Folge von Bytes
  - ◆ beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
  - ◆ dynamisch erweiterbar
  - ◆ Zugriffsrechte: lesbar, schreibbar, ausführbar
- Verzeichnis
  - ◆ baumförmig strukturiert
    - Knoten des Baums sind Verzeichnisse
  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

## 3.1 Pfadnamen

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

### 3.1 Pfadnamen (2)

#### Eigentliche Baumstruktur

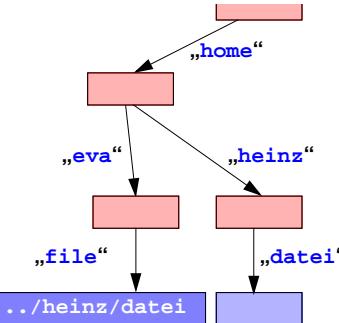


- benannt sind nicht Dateien und Verzeichnisse, sondern die Verbindungen zwischen ihnen
- Verzeichnisse und Dateien können auf verschiedenen Pfaden erreichbar sein z.B. `../heinz/datei` und `/home/heinz/datei`
- Jedes Verzeichnis enthält einen Verweis auf sich selbst (..) und einen Verweis auf das darüberliegende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

### 3.1 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

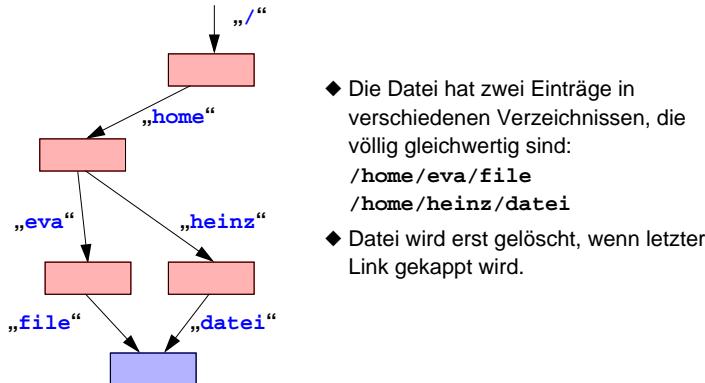


- Symbolischer Name enthält einen neuen Pfadnamen, der vom FS interpretiert wird.

### 3.1 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.

### 3.2 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 Verzeichnis ist genau einem Benutzer und einer Gruppe zugeordnet

#### Rechte auf Dateien

- Lesen, Schreiben, Ausführen (nur vom Eigentümer veränderbar)
- einzelnen 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 erweiterbar“)

### 3.3 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.
- Filedescriptor ist nur prozesslokal gültig.

##### ◆ Sequentialles Lesen und Schreiben

```
ssize_t read( int fd, void *buf, size_t nbytes );
```

Gibt die Anzahl gelesener Zeichen zurück

```
ssize_t write( int fd, void *buf, size_t nbytes );
```

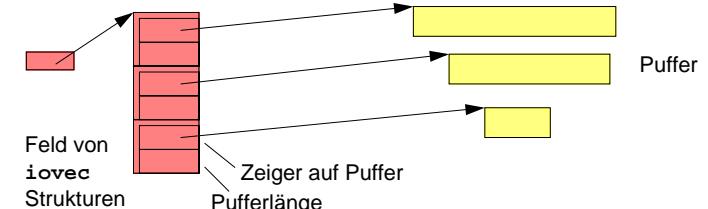
Gibt die Anzahl geschriebener Zeichen zurück

### 3.3 Dateien (2)

#### ■ Weitere Operationen

##### ◆ Lesen und Schreiben in Pufferlisten

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



##### ◆ Positionieren des Schreib-, Lesezeigers

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

### 3.3 Dateien (2)

#### ■ Basisoperationen (2)

##### ◆ Schließen der Datei

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

#### ■ Fehlermeldungen

##### ◆ Anzeige durch Rückgabe von -1

##### ◆ Variable **int errno** enthält Fehlercode

##### ◆ Funktion **perror("")** druckt Fehlermeldung bzgl. **errno** auf die Standard-Ausgabe

### 3.3 Dateien (3)

#### ■ Attribute einstellen

##### ◆ Länge

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

##### ◆ Zugriffs- und Modifikationszeiten

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

##### ◆ Implizite Maskierung von Rechten

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

##### ◆ Eigentümer und Gruppenzugehörigkeit

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

### 3.3 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 );
        Alle Attribute von path ermitteln (folgt symbolischen Links)
int lstat( const char *path, struct stat *buf );
        Wie stat, folgt aber symbolischen Links nicht
int fstat( int fd, struct stat *buf );
        Wie stat, aber auf offene Datei
```

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]

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

C – 21

### 3.4 Verzeichnis

- 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 );
```

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]

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

C – 22

### 3.4 Verzeichnisse (2)

- Verzeichnisse 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 Bibliotheksfunktionen:

```
DIR *opendir( const char *path );
struct dirent *readdir( DIR *dirp );
int closedir( DIR *dirp );
long telldir( DIR *dirp );
void seekdir( DIR *dirp, long loc );
```

- Symbolische Namen auslesen

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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]

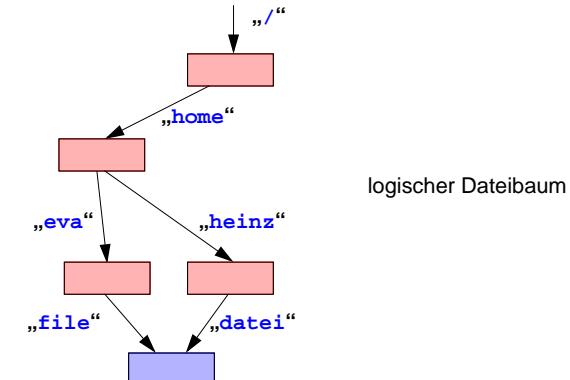
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C – 23

### 3.5 Inodes

- Attribute einer Datei und Ortsinformationen über ihren Inhalt werden in sogenannten Inodes gehalten

- ◆ Inodes werden pro Partition nummeriert (*Inode Number*)



Systemprogrammierung I

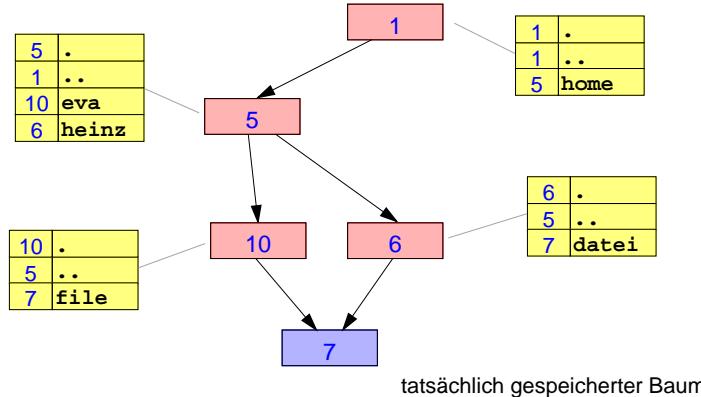
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]

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

C – 24

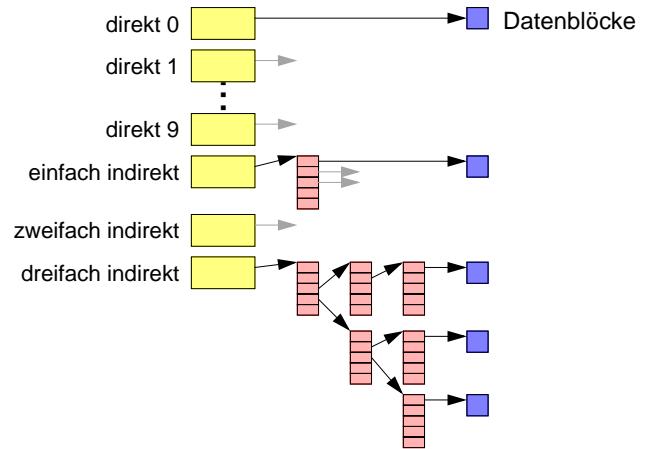
### 3.5 Inodes (2)

- Verzeichnisse enthalten lediglich Paare von Namen und Inode-Nummern



### 3.5 Inodes (4)

- Adressierung der Datenblöcke



### 3.5 Inodes (3)

- Inhalt eines Inodes
  - Inodenummer
  - Dateityp: Verzeichnis, 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 Verzeichnisinhalts (zehn direkte Adressen und drei indirekte)

### 3.6 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

### 3.7 Montieren des Dateibaums

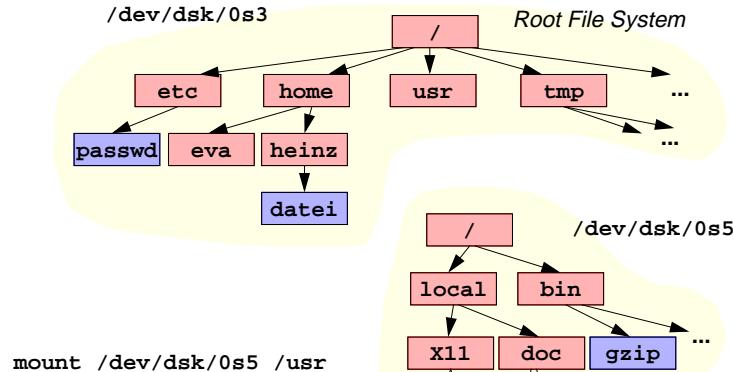
- Der UNIX-Dateibaum kann aus mehreren Partitionen zusammenmontiert werden
    - Partition wird Dateisystem genannt (*File system*)
    - wird durch blockorientierte Spezialdatei repräsentiert (z.B. `/dev/dsk/0s3`)
    - Das Montieren wird *Mounten* genannt
    - Ausgezeichnetes Dateisystem ist das *Root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

Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterrichtsstoffe, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- C - 29

### 3.7 Montieren des Dateibaums (2)

- ## ■ Beispiel

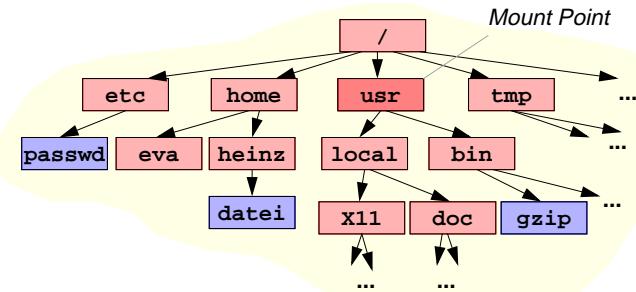


Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10:07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

— C - 30

### 3.7 Montieren des Dateibaums (3)

- ## ■ Beispiel nach Ausführung des Montierbefehls



Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10:07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- C - 31

## 4 Beispiel: Windows 95 (VFAT, FAT32)

- VFAT = Virtual (!) File Allocation Table (oder FAT32)
    - ◆ VFAT: MS-DOS-kompatibles Dateisystem mit Erweiterungen

- Datei
    - ◆ einfache, unstrukturierte Folge von Bytes
    - ◆ beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
    - ◆ dynamisch erweiterbar
    - ◆ Zugriffsrechte: „nur lesbar“, „schreib- und lesebar“

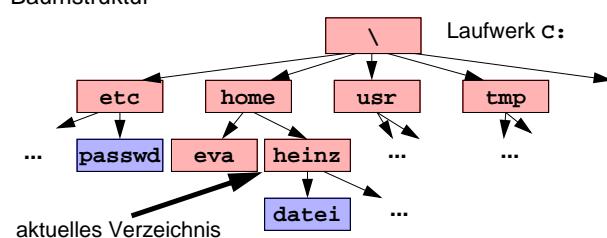
Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[C-File.fm, 2002-10-18 10.07]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- C - 32

## 4 Beispiel: Windows 95 (FAT16, FAT32) (2)

- Partitionen heißen Laufwerke
  - ◆ Sie werden durch einen Buchstaben dargestellt (z.B. c:)
- Verzeichnis
  - ◆ baumförmig strukturiert
    - Knoten des Baums sind Verzeichnisse
    - Blätter des Baums sind Dateien
  - ◆ jedem Windows-Programm ist zu jeder Zeit ein aktuelles Laufwerk und ein aktuelles Verzeichnis pro Laufwerk zugeordnet
  - ◆ Zugriffsrechte: „nur lesbar“, „schreib- und lesbar“

## 4.1 Pfadnamen

- Baumstruktur
- Pfade
  - ◆ z.B. „c:\home\heinz\datei“, „\tmp“, „C:datei“
  - ◆ „\“ ist Trennsymbol (Backslash); beginnender „\“ bezeichnet Wurzelverzeichnis; sonst Beginn implizit mit dem aktuellen Verzeichnis
  - ◆ beginnt der Pfad ohne Laufwerksbuchstabe wird das aktuelle Laufwerk verwendet

## 4.1 Pfadnamen (2)

- Namenskonvention
  - ◆ Kompatibilitätsmodus: 8 Zeichen Name, 3 Zeichen Erweiterung (z.B. AUTOEXEC.BAT)
  - ◆ Sonst: 255 Zeichen inklusive Sonderzeichen (z.B. „Eigene Programme“)
- Verzeichnisse
  - ◆ Jedes Verzeichnis enthält einen Verweis auf sich selbst („..“) und einen Verweis auf das darüberliegende Verzeichnis im Baum („...“) (Ausnahme Wurzelverzeichnis)
  - ◆ keine Hard-Links oder symbolischen Namen

## 4.2 Rechte

- Rechte pro Datei und Verzeichnis
  - ◆ schreib- und lesbar — nur lesbar (read only)
- Keine Benutzeridentifikation
  - ◆ Rechte garantieren keinen Schutz, da von jedermann veränderbar

## 4.3 Dateien

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

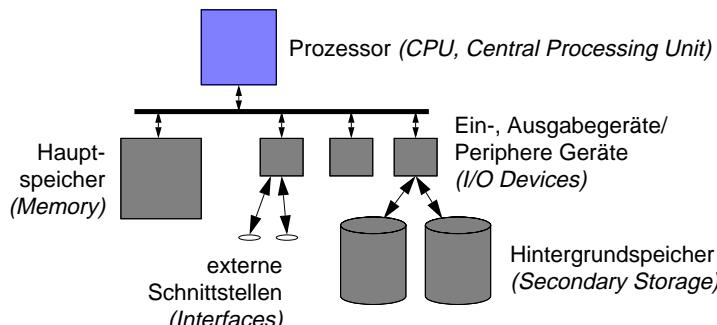
# D Prozesse und Nebenläufigkeit

Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D - 1

## D Prozesse und Nebenläufigkeit

### ■ Einordnung



Systemprogrammierung I  
© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]  
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D - 2

## 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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D - 3

## 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

Systemprogrammierung I

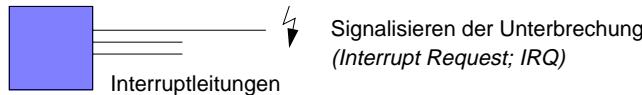
© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D - 4

## 1 Prozessor (3)

### ■ Unterbrechungen (Interrupts)



- ◆ 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

## 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)

## 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 (Beachte: ein Programm kann sich **mehr**fach in Ausführung befinden)

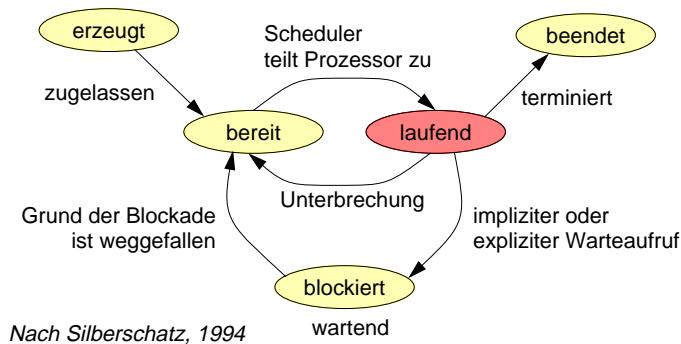
## 2.1 Prozesszustände

### ■ Ein Prozess befindet sich in einem der folgenden Zustände:

- ◆ **Erzeugt (New)**  
Prozess wurde erzeugt, besitzt aber noch nicht alle nötigen Betriebsmittel
- ◆ **Bereit (Ready)**  
Prozess besitzt alle nötigen Betriebsmittel und ist bereit zum Laufen
- ◆ **Laufend (Running)**  
Prozess wird vom realen Prozessor ausgeführt
- ◆ **Blockiert (Blocked/Waiting)**  
Prozess wartet auf ein Ereignis (z.B. Fertigstellung einer Ein- oder Ausgabeoperation, Zuteilung eines Betriebsmittels, Empfang einer Nachricht); zum Warten wird er blockiert
- ◆ **Beendet (Terminated)**  
Prozess ist beendet; einige Betriebsmittel sind jedoch noch nicht freigegeben oder Prozess muss aus anderen Gründen im System verbleiben

## 2.1 Prozesszustände (2)

### Zustandsdiagramm

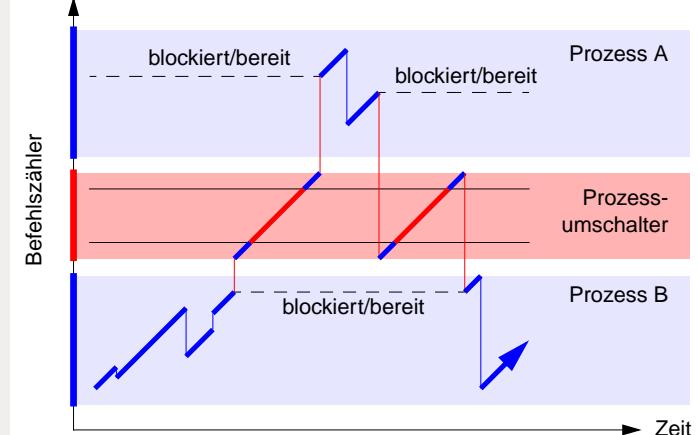


- ◆ Scheduler ist der Teil des Betriebssystems, der die Zuteilung des realen Prozessors vornimmt.

D - 9

## 2.2 Prozesswechsel (2)

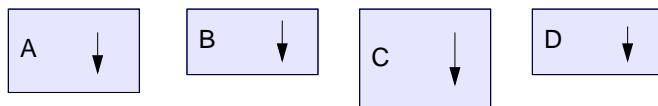
### Umschaltung



D - 11

## 2.2 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.

D - 10

## 2.2 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
  - ...

D - 12

## 2.2 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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D – 13

## 2.3 Prozesserzeugung (UNIX)

- Erzeugen eines neuen UNIX-Prozesses
  - ◆ Duplizieren des gerade laufenden Prozesses

`pid_t fork( void );`

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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D – 14

## 2.3 Prozesserzeugung (UNIX)

- Erzeugen eines neuen UNIX-Prozesses
  - ◆ Duplizieren des gerade laufenden Prozesses

`pid_t fork( void );`

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

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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D – 14

## 2.3 Prozesserzeugung (2)

- ◆ Der Kind-Prozess ist eine perfekte **Kopie** des Vaters
  - Gleiches Programm
  - Gleiche Daten (gleiche Werte in Variablen)
  - Gleicher Programmzähler (nach der Kopie)
  - Gleicher Eigentümer
  - Gleiches aktuelles Verzeichnis
  - Gleiche Dateien geöffnet (selbst Schreib-, Lesezeiger ist gemeinsam)
  - ...
- ◆ Unterschiede:
  - Verschiedene PIDs
  - **fork()** liefert verschiedene Werte als Ergebnis für Vater und Kind

Systemprogrammierung I

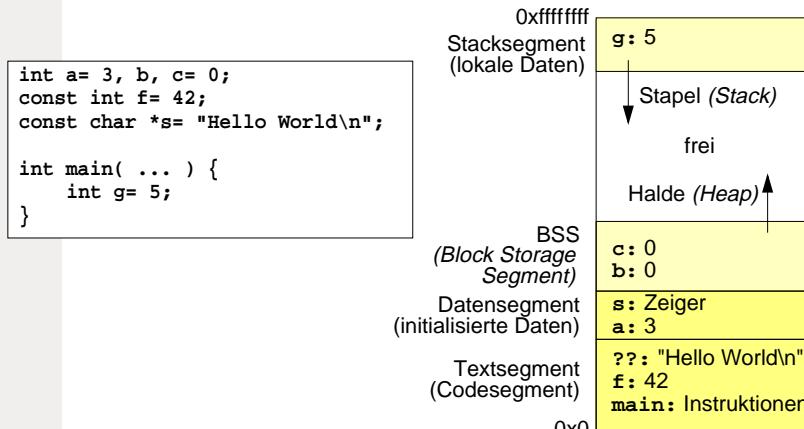
© 1997-2001, Franz J. Hauck, 2000, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[D-Proc.fm, 2002-10-18 10.08]

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

D – 15

## 2.4 Speicheraufbau eines Prozesses (UNIX)

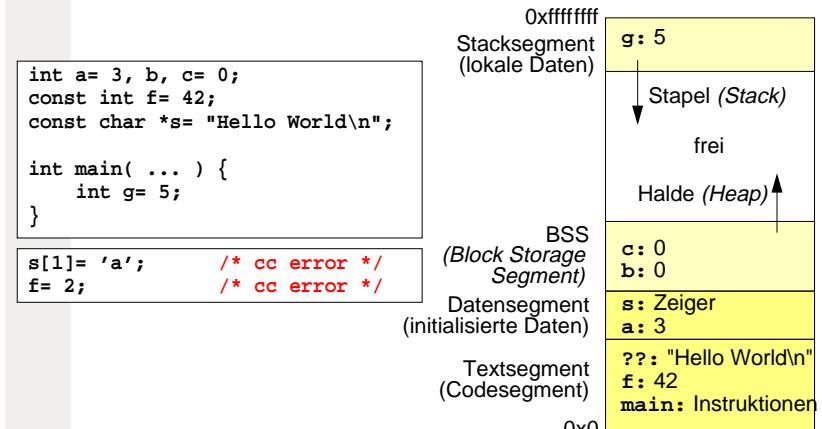
### Aufteilung des Hauptspeichers eines Prozesses in Segmente



D – 16

## 2.4 Speicheraufbau eines Prozesses (UNIX)

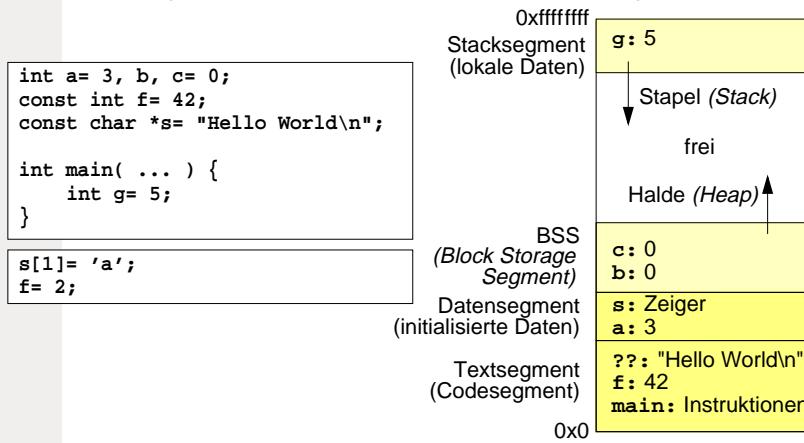
### Aufteilung des Hauptspeichers eines Prozesses in Segmente



D – 16

## 2.4 Speicheraufbau eines Prozesses (UNIX)

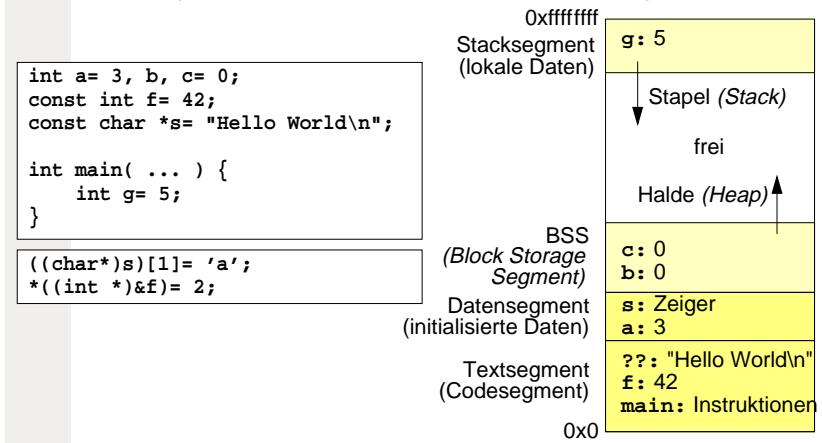
### Aufteilung des Hauptspeichers eines Prozesses in Segmente



D – 16

## 2.4 Speicheraufbau eines Prozesses (UNIX)

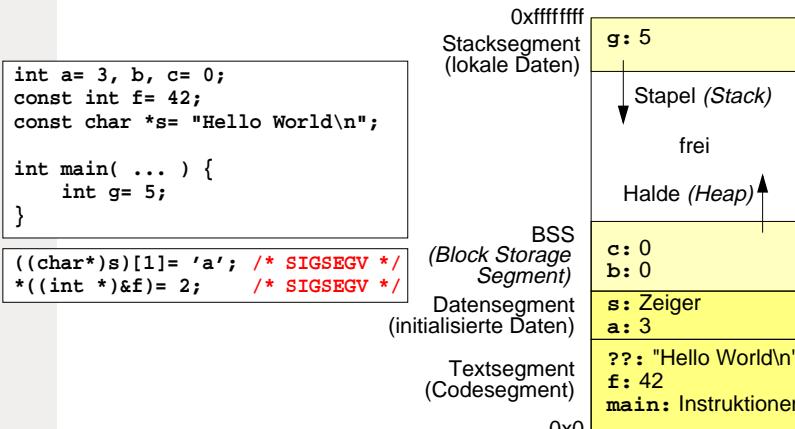
### Aufteilung des Hauptspeichers eines Prozesses in Segmente



D – 16

## 2.4 Speicheraufbau eines Prozesses (UNIX)

- Aufteilung des Hauptspeichers eines Prozesses in Segmente

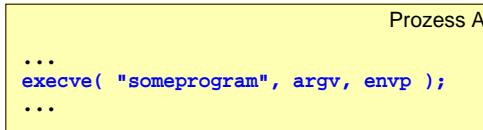


D – 16

## 2.5 Ausführen eines Programms (UNIX)

- Prozess führt ein neues Programm aus

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

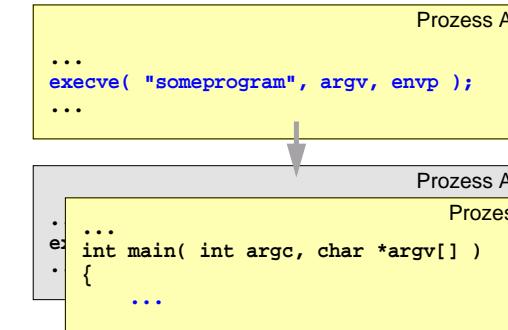


D – 17

## 2.5 Ausführen eines Programms (UNIX)

- Prozess führt ein neues Programm aus

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



Altes ausgeführtes Programm ist endgültig beendet.

D – 17

## 2.6 Operationen auf Prozessen (UNIX)

- Prozess beenden

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

- Prozessidentifikator

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

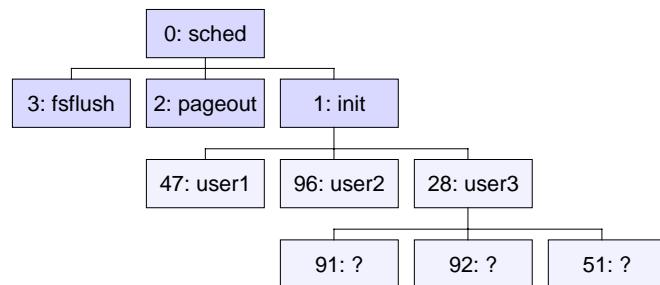
- Warten auf Beendigung eines Kindprozesses

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

D – 18

## 2.7 Prozesshierarchie (Solaris)

- Hierarchie wird durch Vater-Kind-Beziehung erzeugt



Frei nach Silberschatz 1994

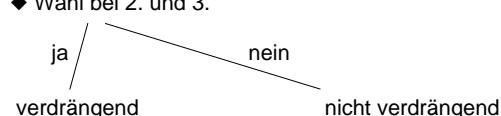
- ◆ Nur der Vater kann auf das Kind warten
- ◆ Init-Prozess adoptiert verwaiste Kinder

D – 19

## 3 Auswahlstrategien

- Strategien zur Auswahl des nächsten Prozesses (*Scheduling Strategies*)

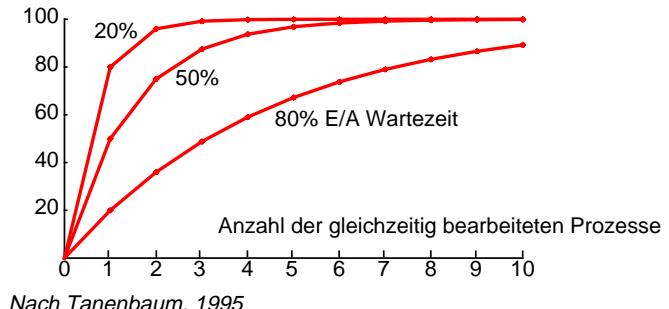
- ◆ Mögliche Stellen zum Treffen von Scheduling-Entscheidungen
  1. Prozess wechselt vom Zustand „laufend“ zum Zustand „blockiert“ (z.B. Ein-, Ausgabeoperation)
  2. Prozess wechselt von „laufend“ nach „bereit“ (z.B. bei einer Unterbrechung des Prozessors)
  3. Prozess wechselt von „blockiert“ nach „bereit“
  4. Prozess terminiert
- ◆ Keine Wahl bei 1. und 4.
- ◆ Wahl bei 2. und 3.



D – 20

## 3 Auswahlstrategien (2)

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



Nach Tanenbaum, 1995

D – 21

## 3 Auswahlstrategien (3)

- Durchsatz
  - ◆ Möglichst hohe Anzahl bearbeiteter Prozesse pro Zeiteinheit
- Verweilzeit
  - ◆ Gesamtzeit des Prozesses in der Rechenanlage soll so gering wie möglich sein
- Wartezeit
  - ◆ Möglichst kurze Gesamtzeit, in der der Prozess im Zustand „bereit“ ist
- Antwortzeit
  - ◆ Möglichst kurze Reaktionszeit des Prozesses im interaktiven Betrieb

D – 22

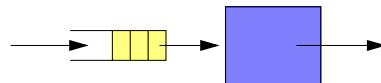
### 3.1 First Come, First Served

- Der erste Prozess wird zuerst bearbeitet (FCFS)

- ◆ „Wer zuerst kommt ...“
- ◆ Nicht-verdrängend

- Warteschlange zum Zustand „bereit“

- ◆ Prozesse werden hinten eingereiht
- ◆ Prozesse werden vorne entnommen



▲ Bewertung

- ◆ fair (?)
- ◆ Wartezeiten nicht minimal
- ◆ nicht für Time-Sharing-Betrieb geeignet

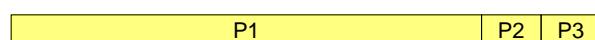
### 3.1 First Come, First Served (2)

- Beispiel zur Betrachtung der Wartezeiten

Prozess 1: 24  
Prozess 2: 3  
Prozess 3: 3

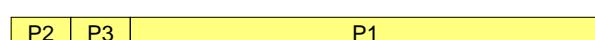
Zeiteinheiten

- ◆ Reihenfolge: P1, P2, P3



mittlere Wartezeit:  $(0+24+27)/3 = 17$

- ◆ Reihenfolge: P2, P3, P1



mittlere Wartezeit:  $(6+0+3)/3 = 3$

### 3.2 Shortest Job First

- Kürzester Job wird ausgewählt (SJF)

- ◆ Länge bezieht sich auf die nächste Rechenphase bis zur nächsten Warteoperation (z.B. Ein-, Ausgabe)

- „bereit“-Warteschlange wird nach Länge der nächsten Rechenphase sortiert

- ◆ Vorhersage der Länge durch Protokollieren der Länge bisheriger Rechenphasen (Mittelwert, exponentielle Glättung)
- ◆ ... Protokollierung der Länge der vorherigen Rechenphase

- SJF optimiert die mittlere Wartezeit

- ◆ Da Länge der Rechenphase in der Regel nicht genau vorhersagbar, nicht ganz optimal.

- Varianten: verdrängend (PSJF) und nicht-verdrängend

### 3.3 Prioritäten

- Prozess mit höchster Priorität wird ausgewählt

- ◆ dynamisch — statisch  
(z.B. SJF: dynamische Vergabe von Prioritäten gemäß Länge der nächsten Rechenphase)  
(z.B. statische Prioritäten in Echtzeitsystemen; Vorhersagbarkeit von Reaktionszeiten)

- ◆ verdrängend — nicht-verdrängend

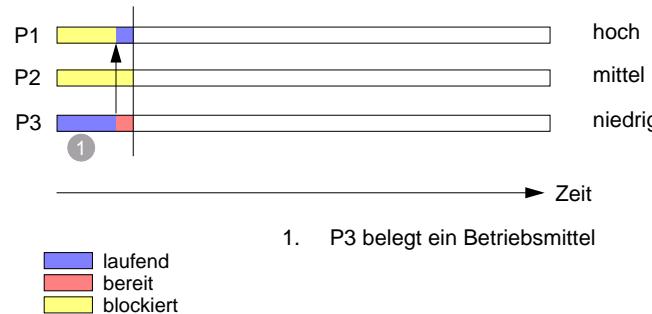
▲ Probleme

- ◆ Aushungerung  
Ein Prozess kommt nie zum Zuge, da immer andere mit höherer Priorität vorhanden sind.
- ◆ Prioritätenumkehr (Priority Inversion)

### 3.3 Prioritäten (2)

#### ■ Prioritätenumkehr

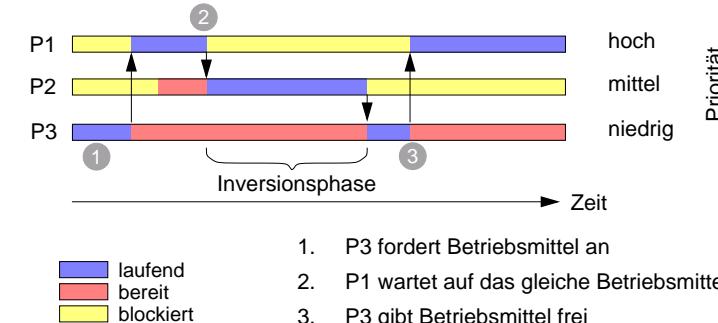
- ◆ hochpriorer Prozess wartet auf ein Betriebsmittel, das ein niedrigpriorer Prozess besitzt; dieser wiederum wird durch einen mittelprioren Prozess verdrängt und kann daher das Betriebsmittel gar nicht freigeben



### 3.3 Prioritäten (2)

#### ■ Prioritätenumkehr

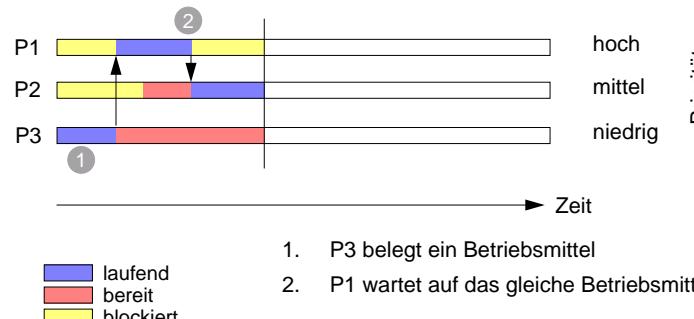
- ◆ hochpriorer Prozess wartet auf ein Betriebsmittel, das ein niedrigpriorer Prozess besitzt; dieser wiederum wird durch einen mittelprioren Prozess verdrängt und kann daher das Betriebsmittel gar nicht freigeben



### 3.3 Prioritäten (2)

#### ■ Prioritätenumkehr

- ◆ hochpriorer Prozess wartet auf ein Betriebsmittel, das ein niedrigpriorer Prozess besitzt; dieser wiederum wird durch einen mittelprioren Prozess verdrängt und kann daher das Betriebsmittel gar nicht freigeben



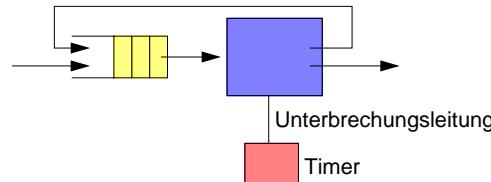
### 3.3 Prioritäten (3)

#### ★ Lösungen

- ◆ zur Prioritätenumkehr:  
dynamische Anhebung der Priorität für kritische Prozesse
- ◆ zur Aushungerung:  
dynamische Anhebung der Priorität für lange wartende Prozesse  
(Alterung, Aging)

## 3.4 Round-Robin Scheduling

- Zuteilung und Auswahl erfolgt reihum
  - ◆ ähnlich FCFS aber mit Verdrängung
  - ◆ Zeitquant (*Time Quantum*) oder Zeitscheibe (*Time Slice*) wird zugewiesen
  - ◆ geeignet für *Time-Sharing*-Betrieb



- ◆ Wartezeit ist jedoch eventuell relativ lang

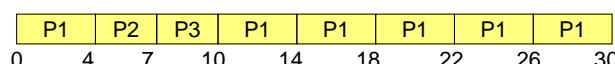
D – 29

## 3.4 Round-Robin Scheduling (2)

- Beispiel zur Betrachtung der Wartezeiten

Prozess 1: 24  
Prozess 2: 3  
Prozess 3: 3 } Zeiteinheiten

- ◆ Zeitquant ist 4 Zeiteinheiten
- ◆ Reihenfolge in der „bereit“-Warteschlange: P1, P2, P3



mittlere Wartezeit:  $(6+4+7)/3 = 5.7$

D – 30

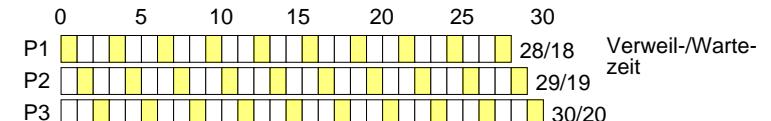
## 3.4 Round-Robin Scheduling (3)

- Effizienz hängt von der Größe der Zeitscheibe ab
  - ◆ kurze Zeitscheiben: Zeit zum Kontextwechsel wird dominant
  - ◆ lange Zeitscheiben: Round Robin nähert sich FCFS an
- Verweilzeit und Wartezeit hängt ebenfalls von der Zeitscheibengröße ab
  - ◆ Beispiel: 3 Prozesse mit je 10 Zeiteinheiten Rechenbedarf
    - Zeitscheibengröße 1
    - Zeitscheibengröße 10

D – 31

## 3.4 Round-Robin Scheduling (4)

- ◆ Zeitscheibengröße 1:



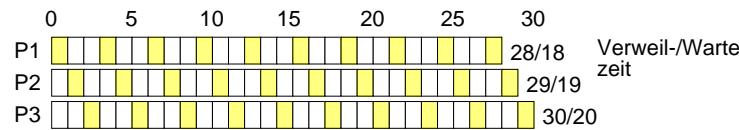
durchschnittliche Verweilzeit: 29 Zeiteinheiten =  $(28+29+30)/3$

durchschnittliche Wartezeit: 19 Zeiteinheiten =  $(18+19+20)/3$

D – 32

### 3.4 Round-Robin Scheduling (4)

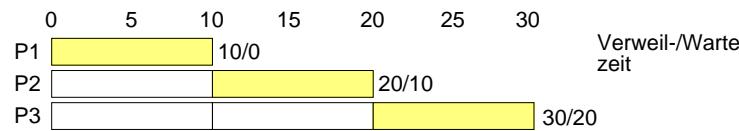
- Zeitscheibengröße 1:



durchschnittliche Verweilzeit: 29 Zeiteinheiten =  $(28+29+30)/3$

durchschnittliche Wartezeit: 19 Zeiteinheiten =  $(18+19+20)/3$

- Zeitscheibengröße 10:



durchschnittliche Verweilzeit: 20 Zeiteinheiten =  $(10+20+30)/3$

durchschnittliche Wartezeit: 10 Zeiteinheiten =  $(0+10+20)/3$

### 3.5 Multilevel-Queue Scheduling

#### ■ Verschiedene Schedulingklassen

- z.B. Hintergrundprozesse (Batch) und Vordergrundprozesse (interaktive Prozesse)
- jede Klasse besitzt ihre eigenen Warteschlangen und verwaltet diese nach einem eigenen Algorithmus
- zwischen den Klassen gibt es ebenfalls ein Schedulingalgorithmus z.B. feste Prioritäten (Vordergrundprozesse immer vor Hintergrundprozessen)

#### ■ Beispiel: Solaris

- Schedulingklassen
  - Systemprozesse
  - Real-Time Prozesse
  - Time-Sharing Prozesse
  - interaktive Prozesse

### 3.5 Multilevel-Queue Scheduling (2)

- Scheduling zwischen den Klassen mit fester Priorität (z.B. Real-Time-Prozesse vor Time-Sharing-Prozessen)

- In jeder Klasse wird ein eigener Algorithmus benutzt:

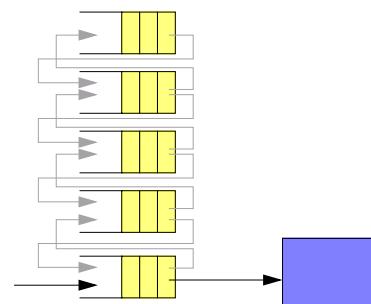
- Systemprozesse: FCFS
- Real-Time Prozesse: statische Prioritäten
- Time-Sharing und interaktive Prozesse: ausgefeiltes Verfahren zur Sicherung von:
  - kurzen Reaktionszeiten
  - fairer Zeitaufteilung zwischen rechenintensiven und I/O-intensiven Prozessen
  - gewisser Benutzersteuerung

#### ★ Multilevel Feedback Queue Scheduling

### 3.6 Multilevel-Feedback-Queue Scheduling

#### ■ Mehrere Warteschlangen (MLFB)

- jede Warteschlange mit eigener Behandlung
- Prozesse können von einer zur anderen Warteschlange transferiert werden



## 3.6 Multilevel Feedback Queue Scheduling (2)

### ■ Beispiel:

- ◆ mehrere Warteschlangen mit Prioritäten (wie bei Multilevel Queue)
- ◆ Prozesse, die lange rechnen, wandern langsam in Warteschlangen mit niedrigerer Priorität (bevorzugt interaktive Prozesse)
- ◆ Prozesse, die lange warten müssen, wandern langsam wieder in höherpriore Warteschlangen (*Aging*)

## 3.7 Beispiel: Time Sharing Scheduling in Solaris

### ■ 60 Warteschlangen, Tabellensteuerung

Level	ts_quantum	ts_tqexp	ts_maxwait	ts_lwait	ts_slpret
0	200	0	0	50	50
1	200	0	0	50	50
2	200	0	0	50	50
3	200	0	0	50	50
4	200	0	0	50	50
5	200	0	0	50	50
...					
44	40	34	0	55	55
45	40	35	0	56	56
46	40	36	0	57	57
47	40	37	0	58	58
48	40	38	0	58	58
49	40	39	0	59	58
50	40	40	0	59	58
51	40	41	0	59	58
52	40	42	0	59	58
53	40	43	0	59	58
54	40	44	0	59	58
55	40	45	0	59	58
56	40	46	0	59	58
57	40	47	0	59	58
58	40	48	0	59	58
59	20	49	32000	59	59

## 3.7 Beispiel: TS Scheduling in Solaris (2)

### ■ Tabelleninhalt

- ◆ kann ausgelesen und gesetzt werden (Auslesen: `dispadmin -c TS -g`)
- ◆ **Level**: Nummer der Warteschlange  
Hohe Nummer = hohe Priorität
- ◆ **ts\_quantum**: maximale Zeitscheibe für den Prozess (in Millisek.)
- ◆ **ts\_tqexp**: Warteschlangennummer, falls der Prozess die Zeitscheibe aufbraucht
- ◆ **ts\_maxwait**: maximale Zeit für den Prozess in der Warteschlange ohne Bedienung (in Sekunden; Minimum ist eine Sekunde)
- ◆ **ts\_lwait**: Warteschlangennummer, falls Prozess zulange in dieser Schlange
- ◆ **ts\_slpret**: Warteschlangennummer für das Wiedereinreihen nach einer blockierenden Aktion

## 3.7 Beispiel: TS Scheduling in Solaris (3)

### ■ Beispielprozess:

- ◆ 1000ms Rechnen am Stück
- ◆ 5 E/A Operationen mit jeweils Rechenzeiten von 1ms dazwischen

#	Warteschlange	Rechenzeit	Prozesswechsel weil ...
1	59	20	Zeitquant abgelaufen
2	49	40	Zeitquant abgelaufen
3	39	80	Zeitquant abgelaufen
4	29	120	Zeitquant abgelaufen
5	19	160	Zeitquant abgelaufen
6	9	200	Zeitquant abgelaufen
7	0	200	Zeitquant abgelaufen
8	0	180	E/A Operation
9	50	1	E/A Operation
10	58	1	E/A Operation
11	58	1	E/A Operation
12	58	1	E/A Operation

### 3.7 Beispiel: TS Scheduling in Solaris (4)

- Tabelle gilt nur unter der folgenden Bedingung:
  - ◆ Prozess läuft fast alleine, andernfalls
    - könnte er durch höherpriore Prozesse verdrängt und/oder ausgebremst werden,
    - wird er bei langem Warten in der Priorität wieder angehoben.

- Beispiel:

#	Warteschlange	Rechenzeit	Prozesswechsel weil ...
...			
6	9	200	Zeitquant abgelaufen
7	0	20	Wartezeit von 1s abgelaufen
8	50	40	Zeitquant abgelaufen
9	40	40	Zeitquant abgelaufen
10	30	80	Zeitquant abgelaufen
11	20	120	Zeitquant abgelaufen
...			

### 3.7 Beispiel: TS Scheduling in Solaris (5)

- Weitere Einflussmöglichkeiten
  - ◆ Anwender und Administratoren können Prioritätenoffsets vergeben
  - ◆ Die Offsets werden auf die Tabellenwerte addiert und ergeben die wirklich verwendete Warteschlange
  - ◆ positive Offsets: Prozess wird bevorzugt
  - ◆ negative Offsets: Prozess wird benachteiligt
  - ◆ Außerdem können obere Schranken angegeben werden
- Systemaufruf
  - ◆ Verändern der eigenen Prozesspriorität

```
int nice( int incr );
```

(positives Inkrement: niedrigere Priorität;  
negatives Inkrement: höhere Priorität)