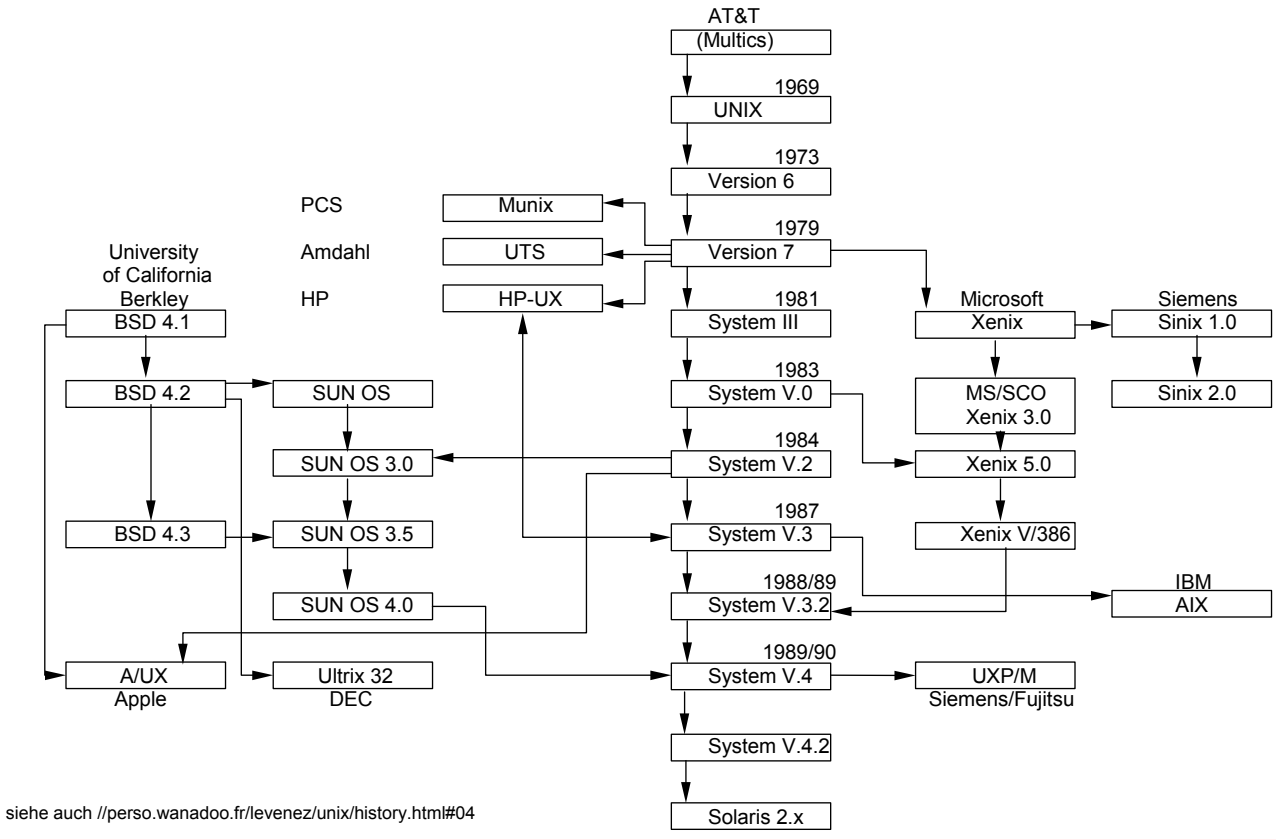
Voraussetzungen

Ziele

Inhalt

- UNIX-Historie
- Betriebssystem-Aufgaben
- Struktur von BS
- BS-Konzepte: Prozesse und Dateien



siehe auch //perso.wanadoo.fr/levenez/unix/history.html#04

M1 BS-Aufgaben (1)

- **Boot**
- BS prüft **Zugriffsberechtigung** (login).
- BS stellt Verbindung zum **Benutzer** her: Kommandointerpretation, Shell, Benutzeroberfläche (Benutzer: Mensch, Programm).
- BS verwaltet **Daten** in Form von Dateien (Files): Zugriff auf Datenfiles auf Massenspeicher (Schreiben, Lesen), Kopieren, Löschen, Benennen, Ordnen. Falls Datei ein ausführbares Programm enthält: BS startet **Ausführung des Programms**.

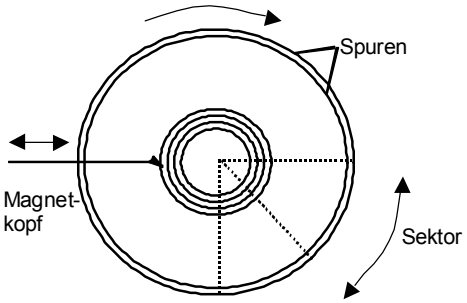
M1 BS-Aufgaben (2)

- BS verwaltet die **Ressourcen**: CPU, Speicherplatz, FD, HD, Netzausgang, Drucker, aber auch SW-Ressourcen: Programme, Prozesse, Tabellen.
- Beispiel Mainframe, 500 Terminals: BS teilt jedem Benutzer (Terminal) die CPU für eine begrenzte Zeit zu. ”Gleichzeitige” Bearbeitung wird vorgetäuscht (**Time Sharing**)
- BS sammelt **Abrechnungsdaten** (bei Mehrbenutzersystemen)
- BS verbirgt Komplexität der HW-Maschine durch Vortäuschung einer komfortableren Maschine (**Abstraktion, Virtualisierung**).

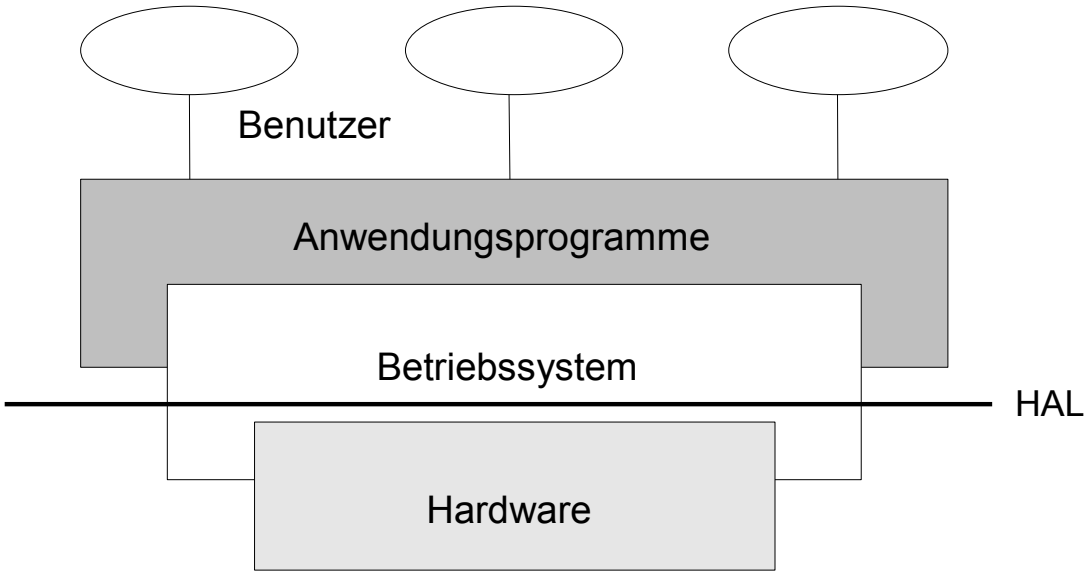
- BS vereinfacht den **Gerätezugriff**. Beispiel: IBM PC Floppy Disk

Anzahl der Zylinder	40	Suchzeit (benachbarter Zylinder)	6 ms
Spuren pro Zylinder	2	Suchzeit (in Mittel)	77 ms
Sektoren pro Spur	9	Umdrehungszeit	200 ms
Sektoren pro Diskette	720	Start- und Stoppzeit des Motors	250 ms
Bytes pro Sektor	512	Übertragungszeit eines Sektors	22 ms
Bytes pro Diskette	368640		

Parameter der IBM PC Disketten



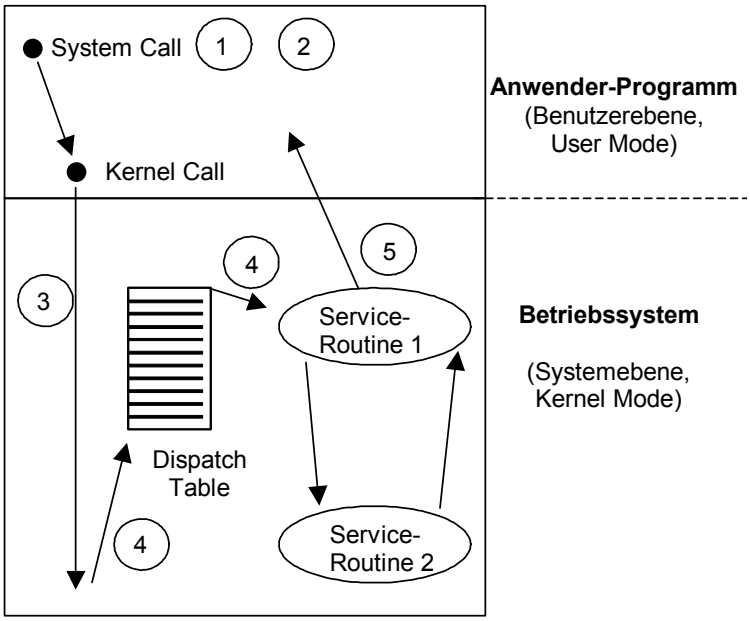
Prinzipbild eines FD-Laufwerks



HAL: Hardware Abstraction Layer

M1 BS-Struktur: Monolithisches Betriebssystem

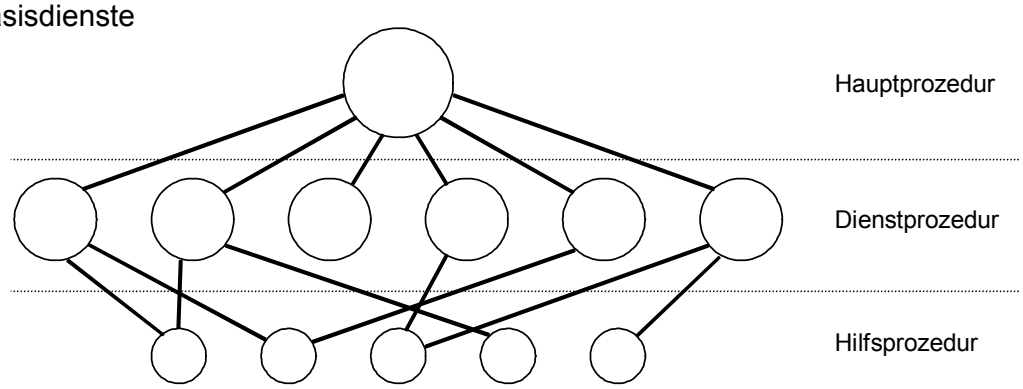
- Aufteilung in **Benutzerebene** (*user*) und **Systemebene** (*kernel*)
- Ablauf:
  1) Anwender-Programm benötigt einen BS-Service: System Call;
  2) Parameter werden in Übergabebereich plaziert;
  3) Steuerung wird an den Systemkern übergeben: Kernel Call (auch: Supervisor Call);
  4) Kernel: identifiziert Service-Routine und ruft sie auf.
  5) Service-Routine läuft ab und gibt Ergebnis an den Auftraggeber (Anwender-Programm) zurück.
- Zweiteilung (user - kernel) ergibt ungenügende Strukturierung.



Struktur eines Monolithischen BS

M1 BS-Struktur: Geschichtetes Betriebssystem

- **3 Ebenen**:
  - User
  - Service Routinen
  - Basisdienste

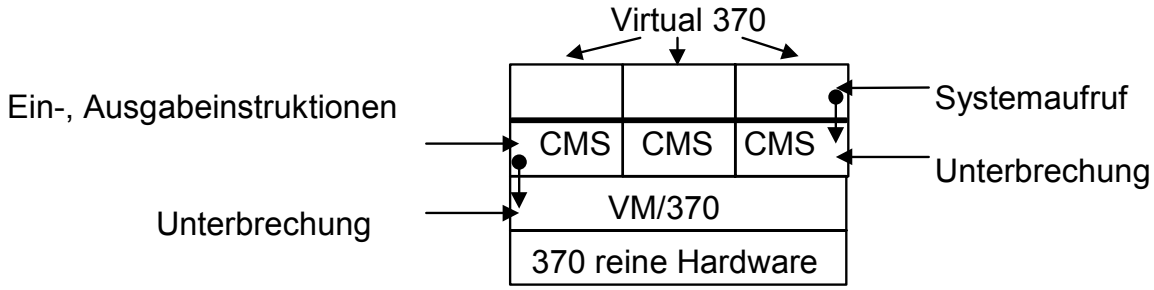


Struktur eines geschichteten BS

- Ebeneneinteilung meist nicht strikt durchgehalten; tendiert zum monolithischen System.

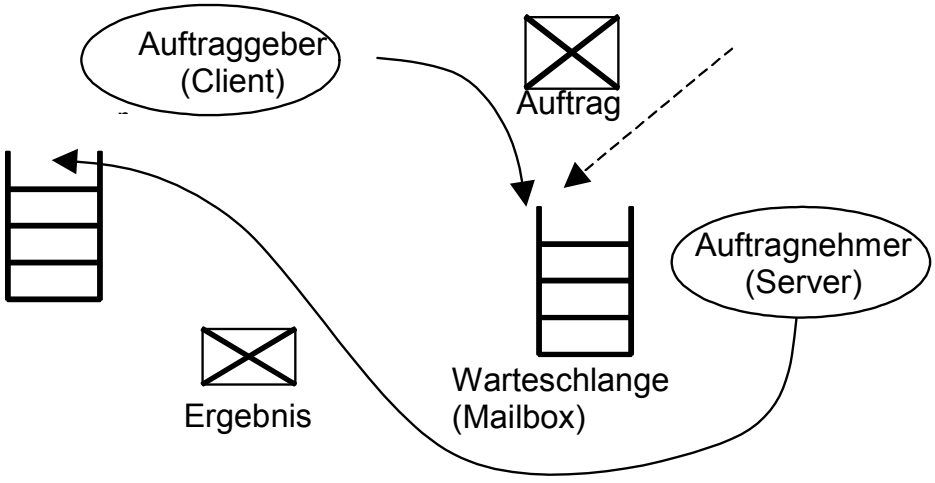
M1 Virtuelle Maschinen

- Entwicklung eines Time-Sharing-Betriebssystems für die IBM/370 führte zu einer Zweiteilung der BS-Aufgaben:
  1) **Mehrfachnutzung einer** Hardware
  2) **Anhebung der HW-Schnittstelle**: extended machine
- zu 1) Virtual Machine Monitor (VM/370) vervielfacht die HW durch exakte Replikation
- zu 2) Auf der vervielfachten HW-Schnittstelle setzt ein (oder auch mehrere verschiedene) Single User-BS auf, z.B. Conversational Monitor System (CMS).



Struktur von VM/370 mit CMS

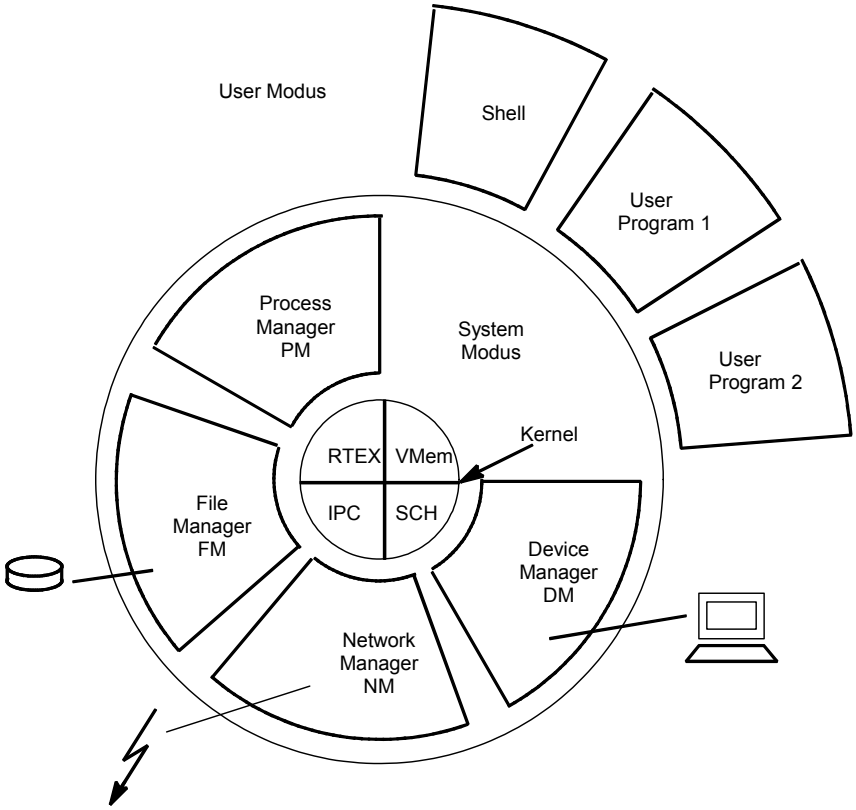
M1 Client-Server-Modell eines BS



Client - Server – Modell eines BS

- Zugrundeliegender Mechanismus: **Nachrichtenaustausch** (*message passing*), Mailboxes, Warteschlangen
- Trend: möglichst kleiner Kernel, der nur die immer nötigen Basisdienste anbietet, Verlagerung der Services in eine höhere (d. h. benutzernähere) Ebene.

M1 Client-Server-Struktur eines BS (UNIX)



M1 Aufgaben des Kernels

- CPU-Verwaltung **(Echtzeitaufgaben, Real Time Executive RTEX) und Scheduling (SCH)**
- **Virtuelle Speicherverwaltung: VMEM**
- Kommunikationsmechanismus: Inter-Prozess-Kommunikation **IPC**
- BS-Dienste werden durch sogen. **Manager** wahrgenommen:
  - File Manager
  - Network Manager
  - Process Manager
  - Terminal Manager
- Systemprogramme (**Utilities**) wandern auf die Anwenderebene:
  - Shell
  - Graf. Benutzeroberfläche
  - Editor
  - Compiler, sowie
  - Anwenderprogramme

M1 Eigenschaften eines Client-Server-BS

- Vorteile
  - modularer Aufbau
  - Kernel-Portierung relativ einfach
  - Austausch oder Weglassen ganzer Module
  - Verteilbarkeit auf mehrere CPUs
  - Trennung in Funktion und Durchführung (Schnittstelle und Implementierung)
- Der Kernel stellt bestimmte **Mechanismen** zur Verfügung, ohne jedoch über deren **Nutzung** (Durchführung) informiert zu sein. Die Manager nutzen den Mechanismus für die Durchführung unterschiedlicher Aufgaben.
- Nachteil:
  - Zeitaufwand für IPC

M1 BS-Konzepte

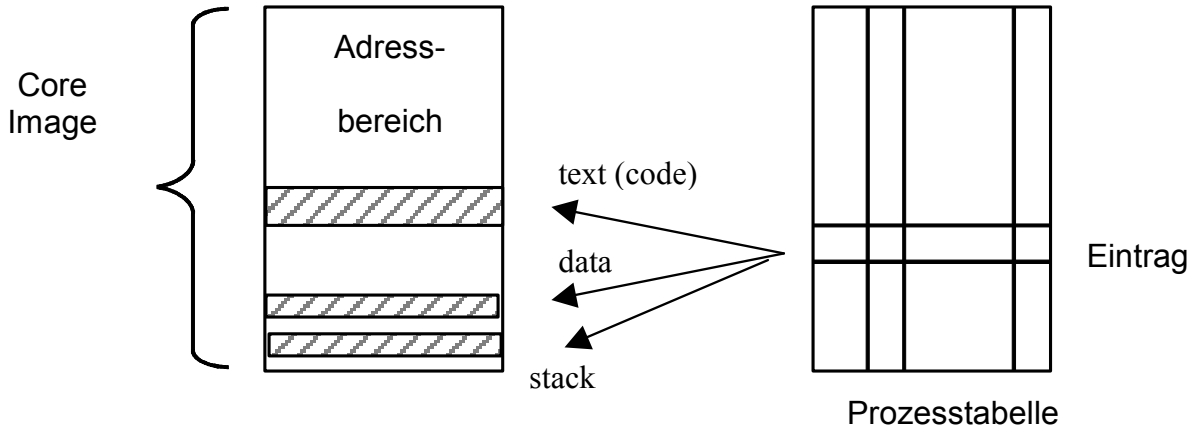
- Das BS bietet dem Anwender (Programm) eine erweiterte Maschinenschnittstelle (**virtuelle Maschine**). Der Befehlssatz der HW ist um die sogen. **System-Aufrufe** (*system calls*) erweitert.
- Die wichtigsten Konzepte (Abstraktionen) eines BS sind:
  - **Prozesse**
  - **Dateien (Files)**

M1 Prozesse

- **Prozess**: ausführbares Programm zusammen mit seiner Umgebung (Prozessor-Zustand, *processor state*)
- **Umgebung**:
  - Program Counter
  - Program Stack, Stack Pointer
  - Data Stack, Stack Pointer
  - Registersatz, evtl. Schattenregister, Flags
  - Filepointer, PID, Priorität etc.
- Bei Programm-Unterbrechung: Rettung der Umgebung so, dass eine spätere Fortsetzung möglich ist.
- Identifikation durch:
  - Process ID: **pid** und
  - User ID: **uid**

M1 Prozess-Tabelle

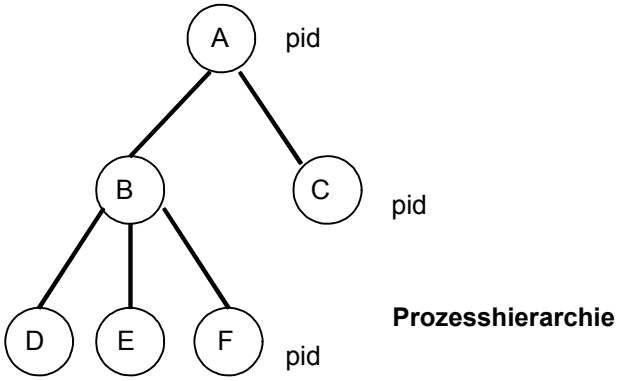
- **Prozess-Tabelle**: Speicherstruktur zur Aufnahme der Umgebung aller Prozesse (sowie anderer Informationen):



Prozess = Core Image + Eintrag in Prozesstabelle

M1 Erzeugen von Prozessen

- Prozesserzeugung durch `pid = fork()`
- `fork` erzeugt einen neuen Prozess (Kind) identisch dem aufrufenden Prozess (Vater)
- Vater und Kind setzen ihre Abarbeitung mit dem auf `fork` folgenden Befehl fort.
- Ein Prozess kann sich mittels `exec` in einen anderen transformieren.



M1 Beenden von Prozessen

- Selbst: `exit(status)`. Im `status` wird dem Vater-Prozess gemeldet, ob der Kind-Prozess ordnungsgemäß beendet wurde.
- Fremd: über **Signale**, z. B. mit dem Shell-Kommando

```
$ kill -9 pid
```

Prompt

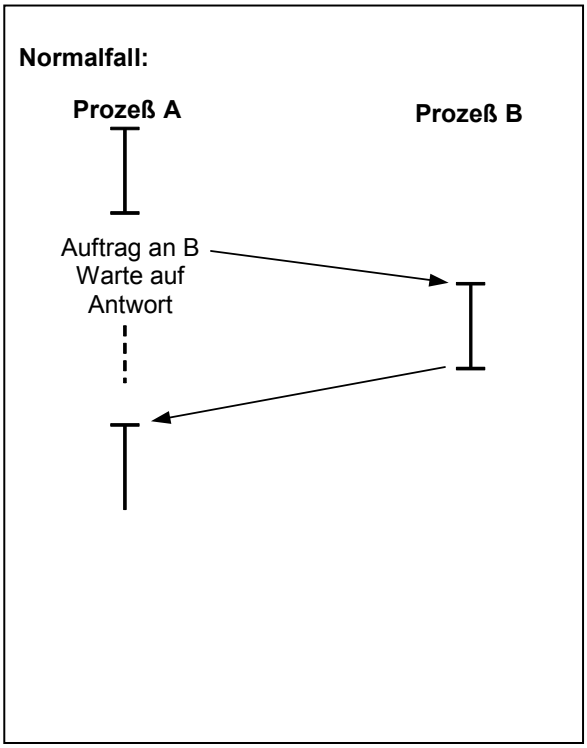
- Anforderung von Speicherplatz: `malloc()`
- Aufgabe von Speicherplatz: `free`
- Versetzen in Wartezustand:

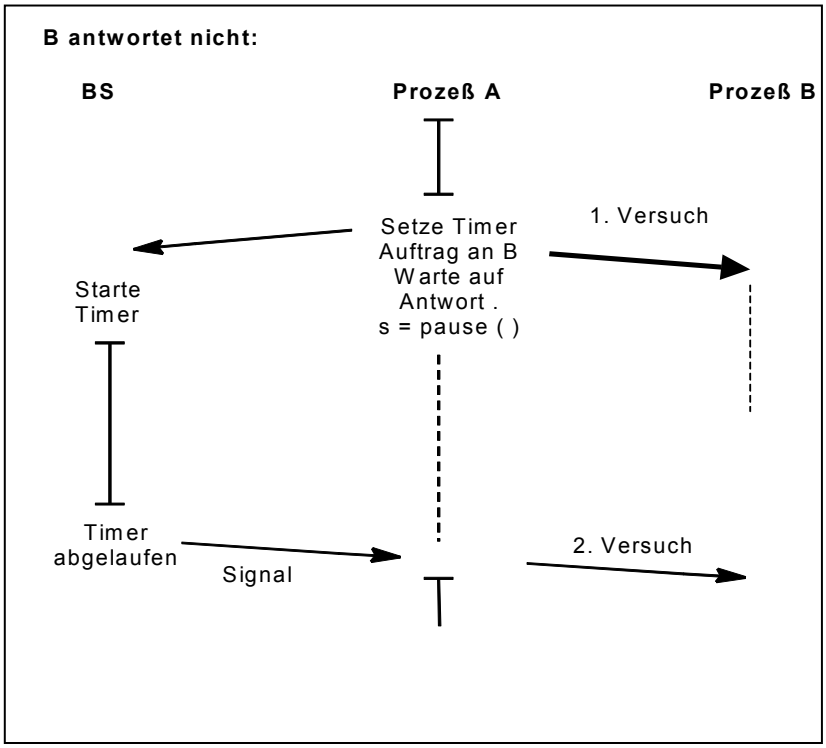
  `s = wait(&status)`

  Warte auf das Ende eines Kind-Prozesses;

  Termination-Status unter der Adresse `&status`.

- **Signale** entsprechen auf SW-Ebene den Interrupts auf HW-Ebene

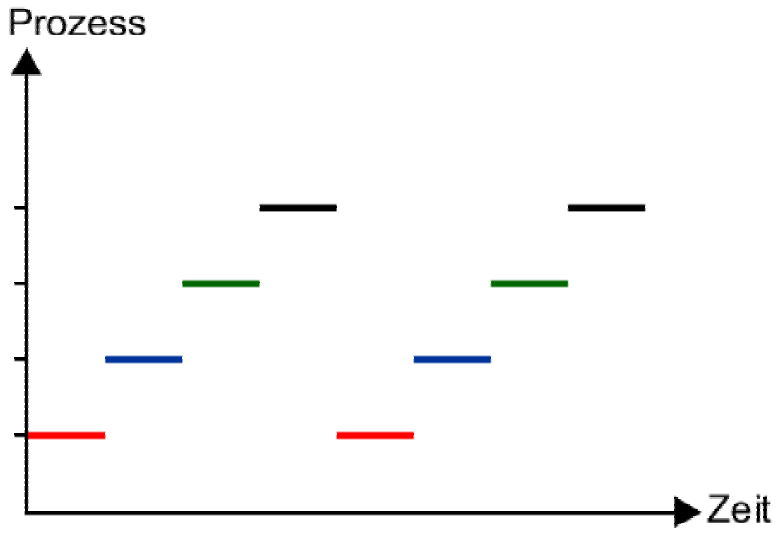




M1 Schutzmechanismen für Prozesse

- Das Recht zur Ausführung eines Prozesses kann in 3 Stufen beschränkt werden:
  - Eigentümer user u
  - Gruppe group g
  - Welt others o
- Der Schutzmechanismus für Prozesse ist **vereinheitlicht** mit dem für Dateien.
- Der **Superuser** hat alle Rechte.
- Ausführungsrechte können **exportiert** werden (Setuid-Mechanismus).

M1 Zeitlicher Ablauf und Prozesszustände



Pseudoparallelität und Realisierung

- Problem: Das exakte zeitliche Verhalten eines Programms ist **nicht voraussagbar**. Echtzeit-Betriebssysteme können allerdings **obere Schranken** für Reaktionszeiten garantieren.

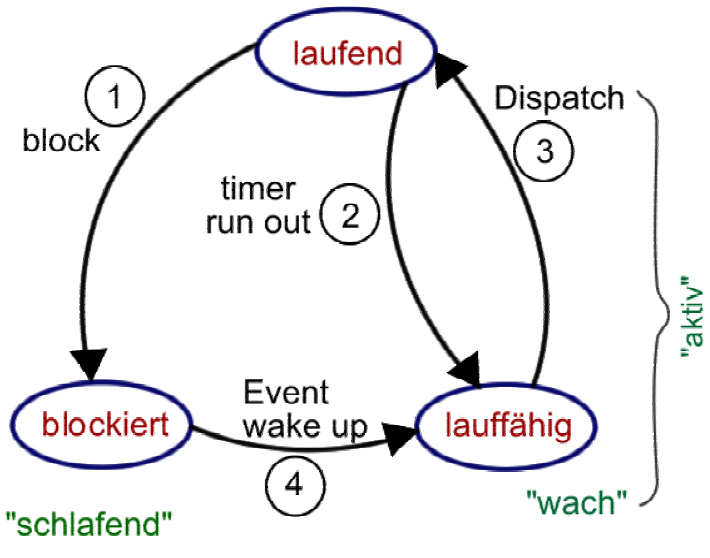
M1 Zustandsübergänge

- Folgende Übergänge zwischen Zuständen sind möglich:

1. Ein laufender Prozess benötigt Daten und kann deshalb nicht weiterlaufen. Die Blockierung erfolgt automatisch oder explizit durch den Prozess selbst (**block**).
2. Ein laufender Prozess wird zwangsweise unterbrochen, um die CPU einem anderen Prozess zuzuteilen (**timer runout**)
3. Die CPU wird einem lauffähigen Prozess zugeteilt (**dispatch**).
4. Der Grund für eine Prozessblockierung ist weggefallen (**wake up**), charakterisiert durch das Eintreffen eines Ereignisses (**event**). Dadurch wird der Prozess zunächst lauffähig, und, falls er die höchste Priorität hat, als nächster zum Ablaufen gebracht (3.).

M1 Zustandsübergangsgraph

- Ein Prozess kann sich in einem von drei **aktiven** Zuständen befinden:
  1. laufend (**running**)
  2. blockiert (**blocked**, z.B. weil er auf Input wartet)
  3. lauffähig (**ready**)



Zustandsübergangsgraph

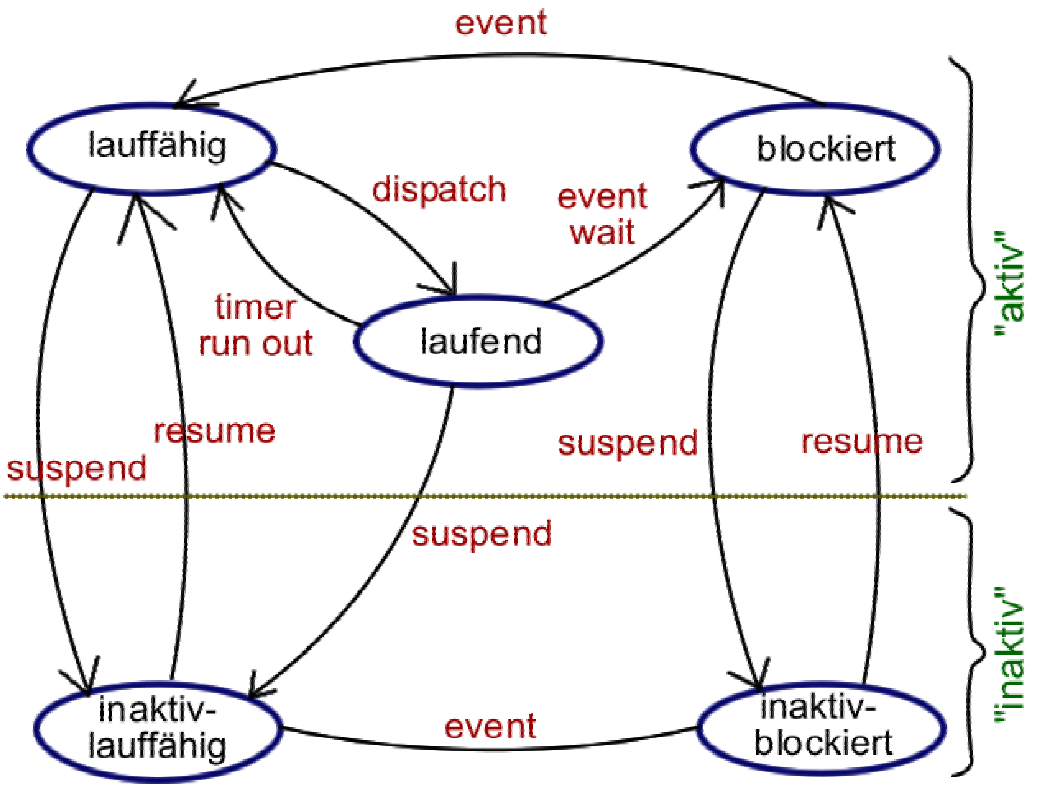
M1 Inaktive Zustände

- Neben den 3 aktiven Zuständen kann ein Prozess auch in einem **inaktiven** (**suspended**) Zustand sein. Aktive Zustände belegen Ressourcen, nicht nur wenn "**laufend**".
- Inaktive Zustände müssen einige ihrer Ressourcen (mindestens: Hauptspeicher) aufgeben.
- Gründe für die Deaktivierung:
  - zeitweise **Überlast** des Systems
  - mögliche **Fehlfunktion** eines Prozesses (Alternative: Abbruch, *abort*)
- Operationen:

  suspend: Übergang aktiv → inaktiv

  resume: Übergang inaktiv → aktiv

M1 Erweiterter Zustandsübergangsgraph

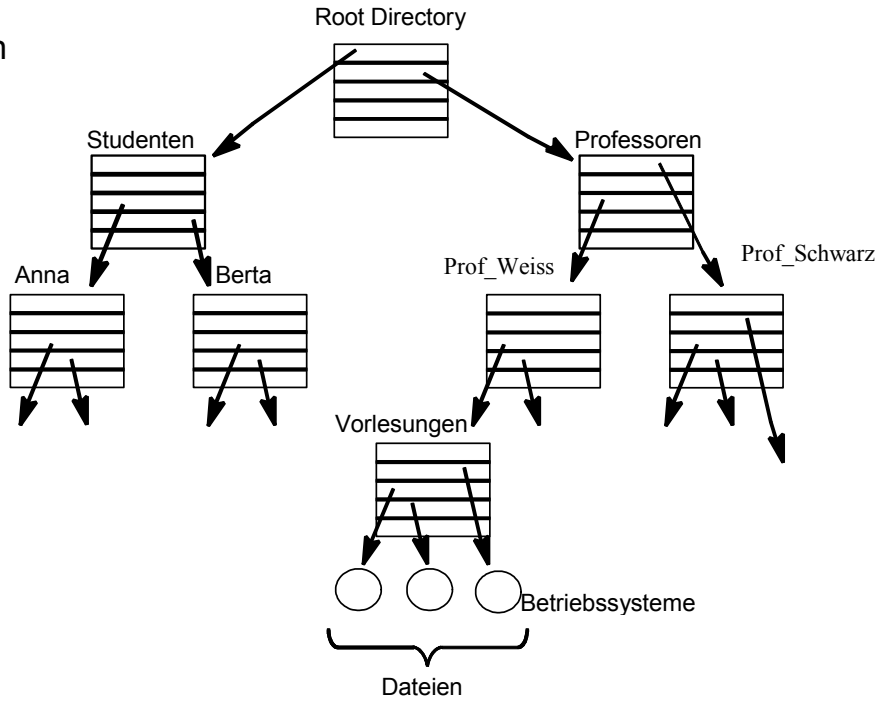


M1 Dateien

- Eine UNIX-Datei ist eine Zeichenfolge (**Bytefolge**).
- Strukturierung durch den Benutzer
- 4 Dateiarten:
  - normale Dateien: Programme, Texte
  - Verzeichnisdateien (Directories): enthalten Verweise auf Dateien und weitere Verzeichnisdateien
  - Gerätedateien (special files)
  - Pipes

M1 Verzeichnisse

- Dateien werden in Verzeichnissen (directories) abgelegt. Directories können weitere Directories enthalten: **Hierarchie**
- Das „höchste“ Directory heißt **Root Directory**.



File-System (File-Baum)

M1 Adressierung von Dateien

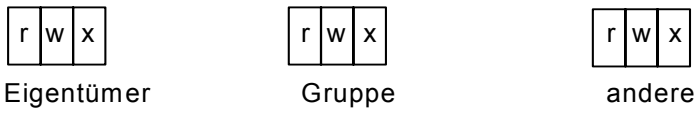
- Durch Angabe ihres **Pfadnamens** (path name), d. h. alle Directories, ausgehend von root, die bis zu der gesuchten Datei durchlaufen werden müssen.
- z.B. /Professoren/Prof_Weiss/Vorlesungen/Betriebssysteme
- **Absolute Pfadnamen** beginnen mit "/", also bei root.
- Genau ein Directory ist jeweils als "**Working Directory**" definiert. **Relative Pfadnamen** beginnen im Working Directory.

  z. B. Working Directory = **`/Professoren`**:

  rel. Pfadname: **`Prof_Weiss/Vorlesungen/Betriebssysteme`**

M1 Schutzmechanismen

- Jede Datei, jedes Directory erhält einen 9-bit-Code (Schutzbits, **Protection Code**):



r: lesen

w: schreiben

x: ausführen (für Verzeichnis: suchen)

- Eine „1" bedeutet: das entsprechende Recht wird gegeben.
- Beispiel: 111 101 001 heißt: rwx r-x --x

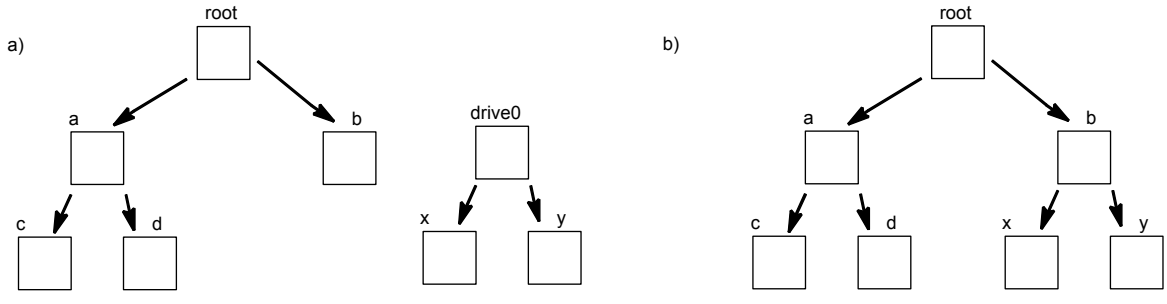
  Eigentümer darf lesen, schreiben, ausführen
  Gruppe darf lesen und ausführen
  alle anderen dürfen nur ausführen.
- Überprüfung der Zugriffsberechtigung beim Öffnen der Datei.

- Das erfolgreiche Öffnen einer Datei resultiert in der Zuteilung eines **File-Deskriptors** (fd).
- Beispiel:

`fd = open(path,flags)` öffnet das durch den Pfadnamen `path` definierte File.

`s = close(fd)` schließt ein geöffnetes File.

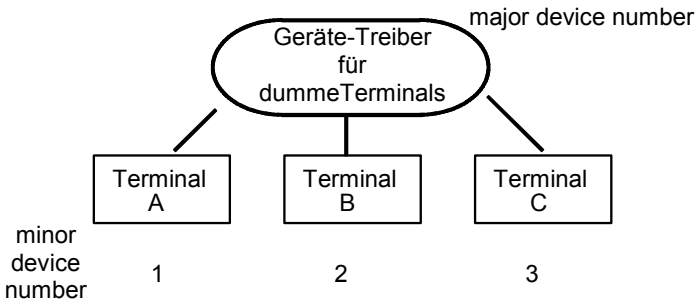
- Problem: Ein entfernbares Speichermedium (Floppy Disk) enthält ein separates File-System. Zugang möglich durch Angabe des absoluten Pfadnamens: `/drive0/x`
- Aber: Konfigurationsabhängigkeit
- Abhilfe: Zweiter File-Baum wird in den ersten eingehängt (”**montiert**”, „**gemountet**“)
- Beispiel: `s = mount(drive0, b, rwflag)` montiert das Block Special File `drive0` unter das Directory `b`.



a) vor dem „Montieren“ b) nach dem „Montieren“

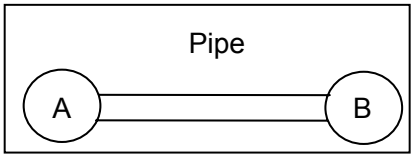
M1 Gerätedateien (special files)

- IO-Geräte (z.B. Terminal, Drucker, Platten, Netz) werden wie Files behandelt (gleiche Systemaufrufe): **Special File**.
- Unterscheidung nach blockorientierten und zeichenorientierten Geräten: **Block Special Files** erlauben wahlfreien (random) Zugriff (z. B. Platte). **Character Special Files** erzeugen bzw. erwarten Zeichenströme (z. B. Terminal, Drucker, Netz-Interface).
- Geräte (devices) werden angesprochen über **Geräte-Treiber** (Device Driver). Sie enthalten (verbergen) die HW-abhängigen Eigenschaften der Geräte.
- Jeweils ein Treiber ist für mehrere gleiche oder ähnliche Geräte (eine Geräteklasse) zuständig. Dem Treiber ist die **Major Device Number** zugeordnet, dem einzelnen Gerät (und damit dem Special File) die **Minor Device Number**.



M1 Pipes

- Eine Datei ist eine **Datenquelle oder -senke**.
- Auch Prozesse können wie Dateien angesprochen werden: sendender Prozess A schreibt in Pseudo-Datei, genannt Pipe, Prozess B liest von Pseudo-Datei.



- Pipe entspricht **FIFO**.
- Beispiel:

```
$ cat file1 file2 file3 | sort  > /dev/lp
```

Die Files 1,2,3 werden conkateniert, sortiert und auf den Lineprinter ausgegeben.