

---

# **BW331**

# **Architektur/Betriebssysteme**

**Vorlesung Hochschule Ludwigshafen  
Sommersemester 2019**

**Dipl.-Inform. Michael Gauglitz  
selbständiger IT-Systemberater und Softwareentwickler  
[michael.gauglitz@its-gauglitz.de](mailto:michael.gauglitz@its-gauglitz.de)**

# Zur Person

---

## Dipl.-Informatiker Michael Gauglitz

Abitur 1990 (Ludwigshafen)

Studium an der Universität Kaiserslautern  
Informatik, Nebenfach Mathematik  
Abschluss 1996

ONSYS GmbH, Ludwigshafen (1996-2004)

Systemadministrator Unix, Datenbanken, Applikationsserver

Softwareentwickler Datenbanken, Data Warehouse, Automatisierungstechnik

seit 2004 IT Services Gauglitz

selbständiger IT-Systemberater und Softwareentwickler

seit Sommersemester 2005 Vorlesung Architektur/Betriebssysteme

[michael.gauglitz@its-gauglitz.de](mailto:michael.gauglitz@its-gauglitz.de)

# **IT Services Gauglitz**

---

Erbringung von IT-Dienstleistungen

Schwerpunkte:

Betriebssysteme (Unix, Linux)

Datenbanken, Data Warehouse (Oracle)

Softwareentwicklung SQL, PL/SQL, Visual Basic, C, C++, C#

Softwarevalidierung

Branchen-Knowhow:

Chemie, Medizintechnik, Automatisierungstechnik, Lebensmittelindustrie

# Organisatorisches

---

Vorlesung und Übung:

freitags 08:30 – 11:45

Raum E1106, 05.04., 12.04., 26.04., 03.05., 24.05., 07.06., 14.06.

Linux-Accounts werden für alle angelegt, die sich in die Umlaufliste in der ersten Vorlesung eintragen

Lösungen der Übungsblätter werden nach Vorlesungs- und Übungsfortschritt online zur Verfügung gestellt

PDF der Vorlesungsfolien und Übungsblätter passwortgeschützt zum Download unter:

<http://www.its-gaуглitz.de/fh>

# Ziel der Vorlesung

---

- Vermittlung von Grundlagen und Konzepten von Betriebssystemen im allgemeinen, die sich in allen Betriebssystemen wieder finden
- Umsetzung dieser Konzepte anhand des Betriebssystems "UNIX" bzw. "Linux"
- Erlernen der Grundlagen zum praktischen Umgang mit dem Betriebssystem "Linux"
- Ihr erfolgreiches Bestehen der Abschlussklausur

# Motivation

---

- Um erfolgreich Anwendungssoftware entwickeln zu können, die effizient und stabil ist, muss man die Umgebung kennen, in der die Software eingesetzt wird.
- Neben der Hardware stellen Betriebssysteme den wichtigsten Teil dieser Umgebung dar.

# Inhaltsübersicht

---

- 1. Betriebssystemdefinition und Rechnerarchitektur**
- 2. Erste Schritte auf einem UNIX-System**
- 3. UNIX-Shells: Grundlagen**
- 4. Prozessmanagement**
- 5. Speichermanagement**
- 6. Input/Output-Management**
- 7. Dateisysteme**
- 8. UNIX-Tools**

# Literaturhinweise

---

## Betriebssysteme allgemein

[1] Andrew S. Tanenbaum: Moderne Betriebssysteme, 2. Auflage,  
Pearson Studium, 2002, 3. aktualisierte Auflage 2009,  
4. aktualisierte Auflage 2016  
ISBN 3-8273-7019-1, 59,95€

## Unix / Linux

- [2] Matthias Mann: Das Einsteigerseminar UNIX, vmi-Buch,  
ISBN 3-8266-7198-8, 9,95€
- [3] Ellen Siever u.a., Linux in a Nutshell, O'Reilly, ISBN 3-89721-195-5, 36€

## Abbildungen

Die meisten Abbildungen stammen vom Autor des Buches [1].

# **Kapitel 1- Betriebssystemdefinition und Rechnerarchitektur**

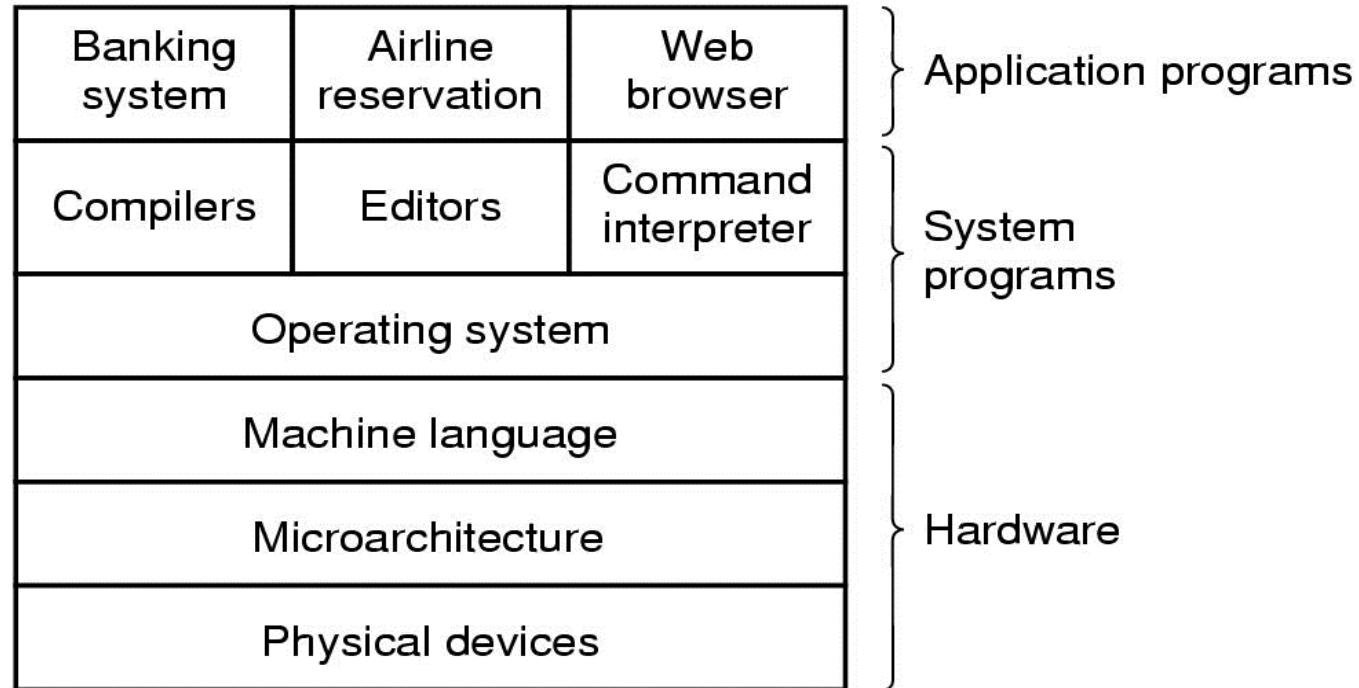
---

**1.1 Betriebssystem: Definition**

**1.2 Wichtige Hardwarekomponenten**

**1.3 Das UNIX-Betriebssystem**

# 1.1 Betriebssystem: Definition



Funktionale Sicht eines Computersystems

- Hardware
- Systemprogramme
- Applikationen

# 1.1 Betriebssystem: Definition

---

**Was ist ein Betriebssystem (auch OS = Operating System) ?**

## virtuelle Maschine

- Abstraktion von der konkreten Hardware, welche einfacher zu programmieren ist
- konkrete Hardware-Details werden verborgen

Aufgaben:

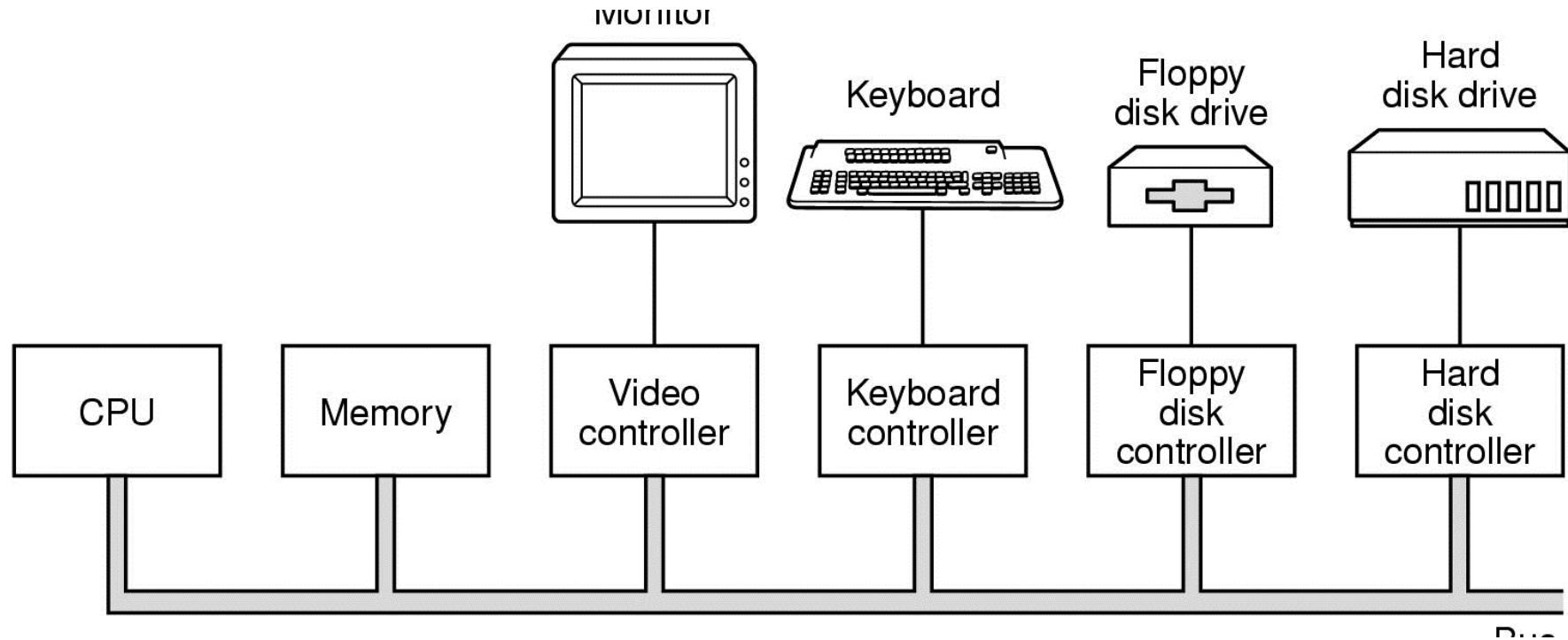
- Bereitstellung von Benutzer-, Programmierschnittstellen (GUI, CLI, API)

## Ressourcenmanager

Aufgaben:

- effiziente Verwaltung der Betriebsmittel eines Rechners
- zeitliche Ressourcen multiplexen: Bsp. CPU, Drucker
- räumliche Ressourcen multiplexen: Bsp. Hauptspeicher, Festplattenplatz

# 1.2 Wichtige Hardwarekomponenten



Einfacher Personalcomputer

# 1.2 Wichtige Hardwarekomponenten

---

## 1.2.1 CPU (Central Processing Unit)

- Gehirn des Computers
- Bestandteile der CPU:
  - Befehlssatz: einfache Operationen (OR, AND, ...), Lade- und Speicheroperationen, Sprünge, Unterprogramme
  - Registersatz: Speicherzellen für Programmzähler, Stack Pointer, PSW (Program Status Word)
  - Steuerwerk: Laden, Dekodierung, Interpretation der Befehle, Erzeugung der Steuersignale zur Ausführung der Befehle
  - Rechenwerk: Ausführung von Rechenoperationen (arithmetische und logische Verknüpfungen), auch ALU genannt

# 1.2 Wichtige Hardwarekomponenten

---

## Mehrkernprozessoren (Multi-Core):

- Mikroprozessor mit mehr als einem Hauptprozessor auf einem Chip
- alle Ressourcen (z.B. ALU, Registersätze etc.) sind mehrfach vorhanden (Ausnahme: Bus, einige Caches)
- Dual Core (zwei Prozessorkerne), Quad Core (vier Prozessorkerne)
- früher: Leistungssteigerung in der Regel durch neue Befehle, mehr Transistoren, höhere Taktrate, bei Frequenzen um 4GHz wird nicht mehr handhabbare Abwärme erzeugt
- deshalb seit 2006 Dual Core, seit 2007 auch Quad-Core

Anderer Ansatz:

Multi threaded CPUs: eine CPU mit mehreren Programmzählern und Registersätzen, melden sich im System als mehrere Kerne

Beispiele für Dual Core Prozessoren:

AMD Athlon 64 FX, Athlon X2, HP PA-8800, IBM POWER5+, Intel Core 2 Duo, SUN UltraSPARC IV+

Beispiele für Quad Core Prozessoren:

AMD Opteron, IBM POWER 6, Intel Core 2 Quad, SUN UltraSPARC T1

seit etwa 2010 Prozessoren mit 6 (Hexacore) und 8 (Octacore) Kernen von Intel und AMD

# 1.2 Wichtige Hardwarekomponenten

---

## Zwei Betriebsmodi der CPU:

### Usermode

- auch unprivilegiert genannt
- kein Hardwarezugriff möglich
- Applikationen laufen in diesem Modus

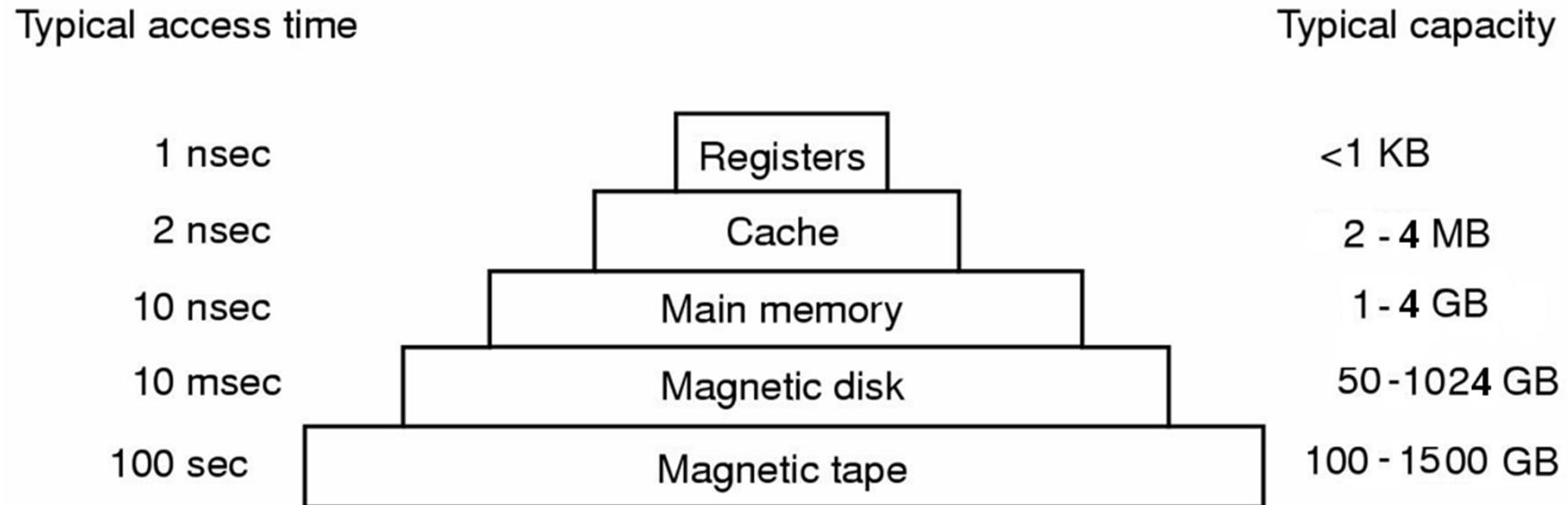
### Kernelmode

- auch privilegiert genannt
- Hardwarezugriff möglich
- Betriebssystem läuft in diesem Modus

# 1.2 Wichtige Hardwarekomponenten

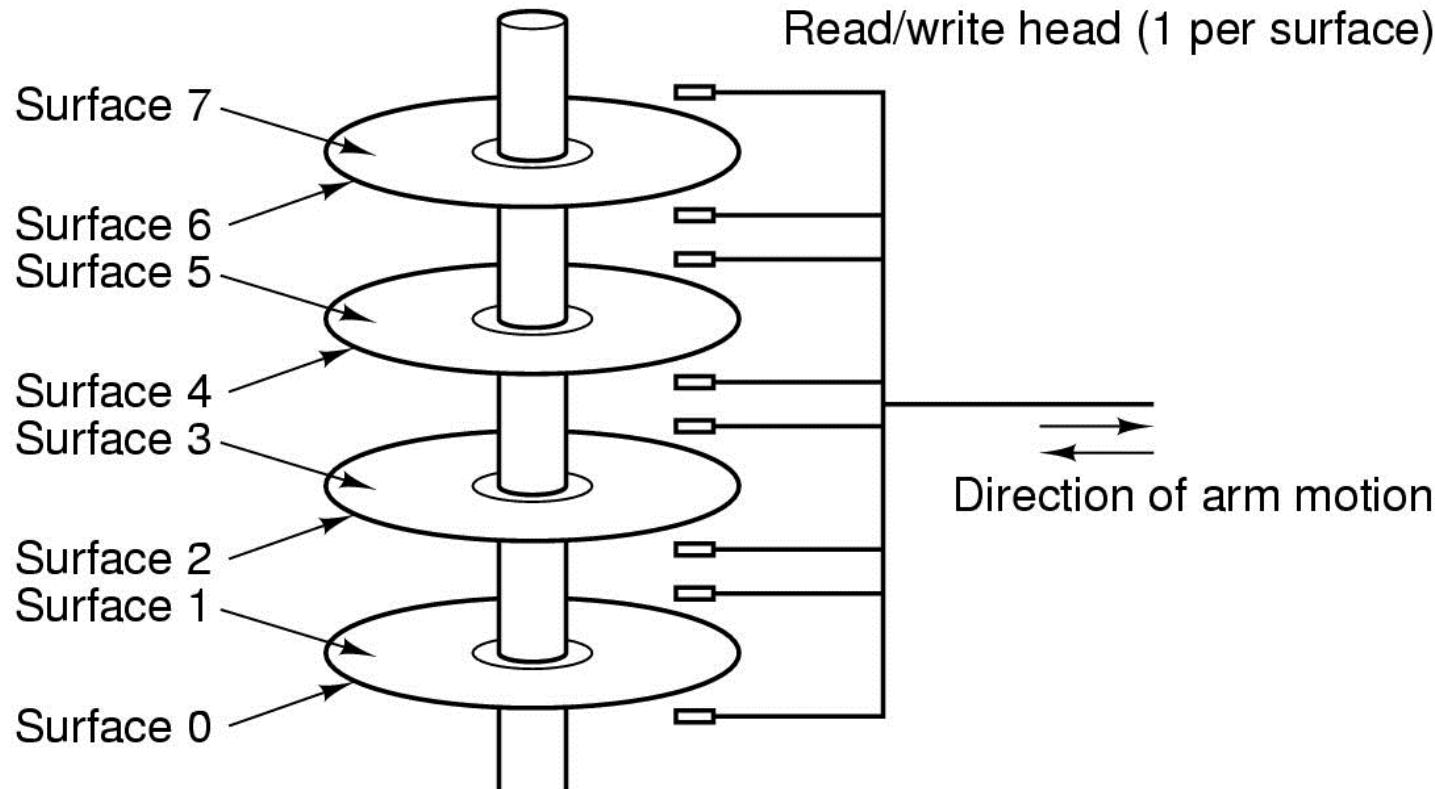
## 1.2.2 Speicher

- ideal: sehr schnell, sehr groß, sehr billig (nicht realisierbar !!!)
- deswegen: Speicherhierarchie (Kosten / Bit sinken mit der Kapazität)



# 1.2 Wichtige Hardwarekomponenten

## Aufbau einer Festplatte



Größen und Speicherkapazitäten: 3.5" (bis 6 TB), 2.5" (bis 2 TB) und 1.8" (bis 320 GB)

# 1.2 Wichtige Hardwarekomponenten

---

weitere Speicherarten:

- ROM (Read Only Memory): z.B. beim Booten benutzt
- EEPROM (Electrically Erasable ROM), Bsp. Teil des BIOS (Basic Input Output System)
- CMOS-RAM (Non volatile RAM, z.B. Uhr des Computers, batteriegespeist), CMOS ist Schaltungstechnik: Complementary Metal Oxide Semiconductor
- Flash-Speicher (eigentlich Flash-EEPROM): Bytes nicht einzeln lösbar
- optische Speicher (CD-ROM, CD-RW, DVD-ROM, DVD-RW, Blu-Ray)

# 1.2 Wichtige Hardwarekomponenten

---

## 1.2.3 Ein- / Ausgabegeräte

- bestehen in der Regel aus zwei Komponenten:  
Controller (Chip oder Platine) und Gerät (Device)
  - über Gerätetreiber (Device Driver) im Kernel spricht OS Controllerschnittstelle an
  - Controller steuert Gerät (Bsp. Plattenkopf)

# 1.2 Wichtige Hardwarekomponenten

---

## 1.2.4 Bussysteme

dienen dem Datentransfer zwischen den Rechnerkomponenten

Klassifikation von Bussen:

- Breite: Anzahl der Verbindungsleitungen
- Funktion: Adressen, Daten, Befehle, Steuersignale
- Betriebsart: unifunktional, multifunktional (Adressen und Daten), seriell, parallel
- Taktung: Geschwindigkeit, wie schnell Daten übertragen werden können

Faustformel:

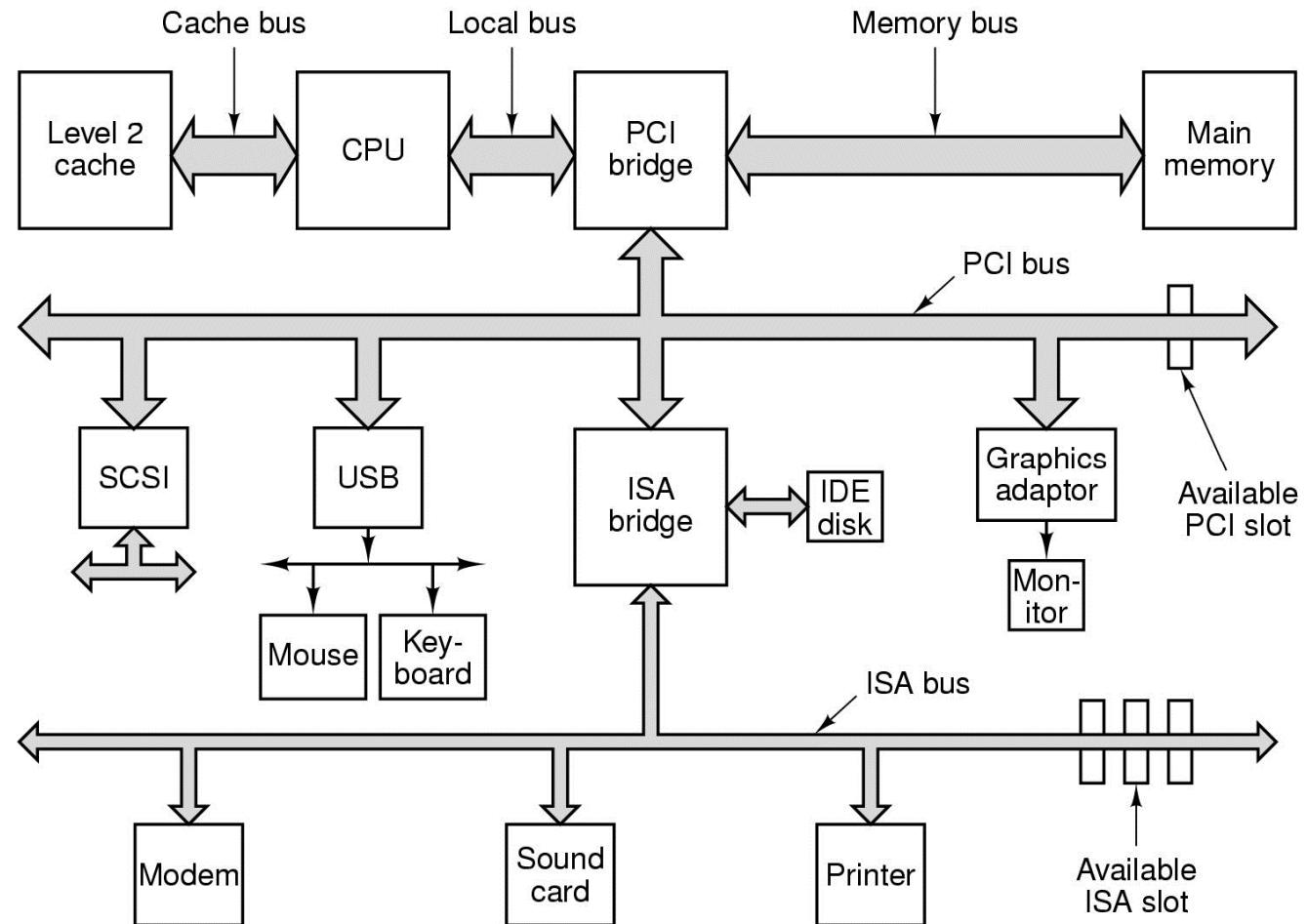
$$\text{Übertragungsrate} = \text{Busbreite} * \text{Bustakt}$$

z.B. bei 16 Bit Busbreite und 10 MHz Bustakt ergeben 20 MB/s  
Übertragungsrate

# 1.2 Wichtige Hardwarekomponenten

Weil CPU und Speicher immer schneller werden, umfaßt ein modernes System mehrere Busse.

Bsp.: Pentium PC 1997



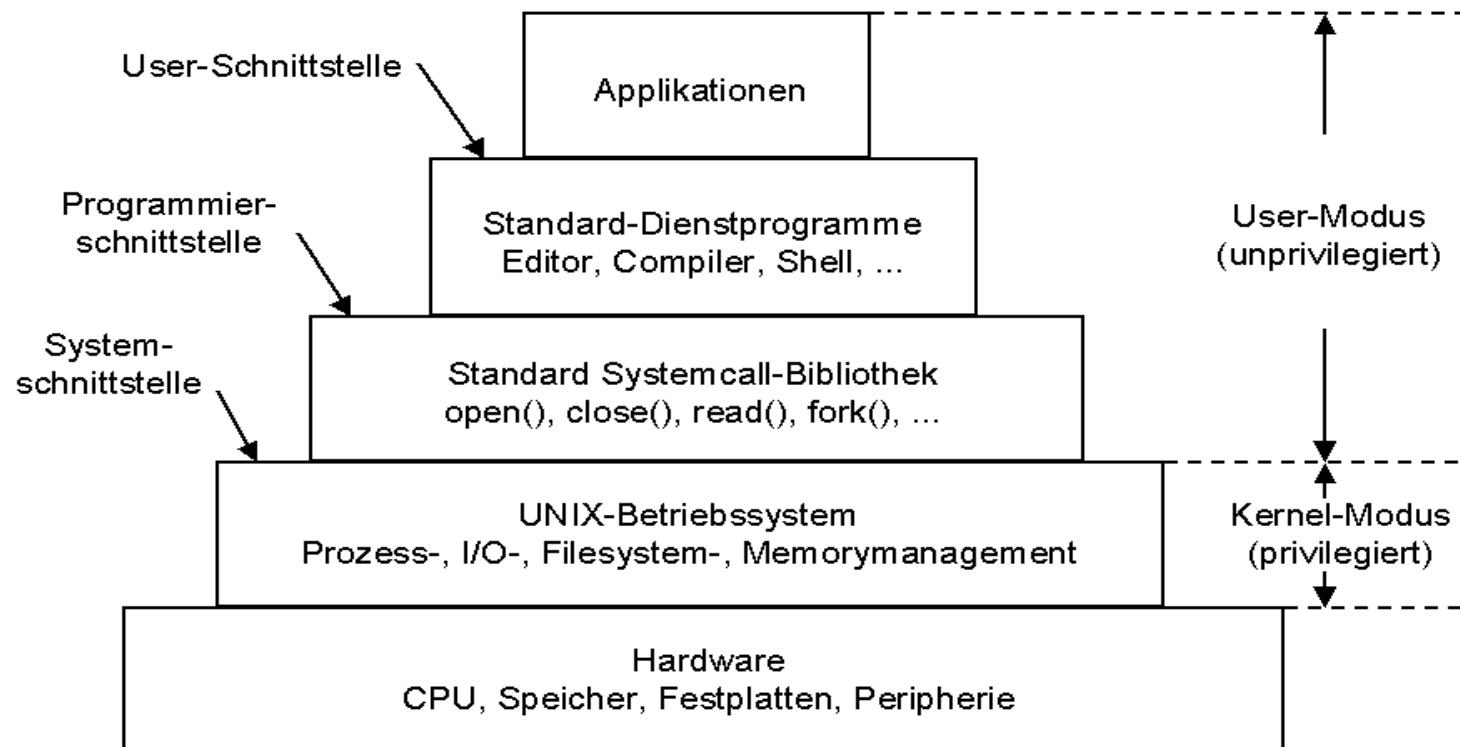
## 1.2 Wichtige Hardwarekomponenten

---

- ISA: Industry Standard Architecture (16 Bit, 8.33 MHz, 16.67 MB/s)
- PCI: Peripheral Component Interconnect (66 MHz, 16 Bit 132 MB/s, 32 Bit 264 MB/s, 64 Bit 528 MB/s)
- Speicherbus: früher: 100/133 MHz (doublepumped, 2 Datenpakete pro Takt), heute 333/400 MHz (quadpumped, 4 Datenpaket pro Takt)
- USB: Universal Serial Bus, ein Treiber für alle Geräte (USB 1.1 Full Speed 1.5 MB/s, USB 2.0 60 MB/s, spezielle Schirmung, USB 3.0 geplant ca. 500 MB/s, Lichtwellenleiter)
- SCSI: Small Computer System Interface (für Massenspeicher), bis zu 320 MB/s
- IEEE 1394 (FireWire): 50 MB/s, für Multimediasysteme
- ATA/ATAPI (IDE): für Massenspeicher wie Festplatte und CD/DVD-Laufwerk, von 2,1 MB/s bis 133 MB/s
- Serial ATA: Weiterentwicklung von ATA/IDE, 150 MB/s bis 300 MB/s, 2008: Serial ATA 6,0 Gbit/s, 2013: SATA Express 8 und 16 Gbit/s

# 1.3 Das UNIX-Betriebssystem

## Das UNIX-Schichtenmodell



# Zusammenfassung Kapitel 1

---

- Der Begriff Betriebssystem kann auf zwei Arten definiert werden: als virtuelle Maschine und als Ressourcenmanager. Die Aufgaben eines Betriebssystems bestehen darin, Benutzer- und Programmierschnittstellen zur Verfügung zu stellen und die Betriebsmittel eines Rechners effizient zu verwalten.
- Die CPU ist das "Gehirn" des Computers. Die CPU besteht aus dem Steuerwerk, dem Rechenwerk und hat einen Befehls- und Registersatz. Die CPU arbeitet in zwei Betriebsmodi.
- Da es nicht möglich ist, sehr schnellen und sehr großen Speicher sehr kostengünstig zu entwickeln, gibt es eine Speicherhierarchie, in der die Kosten/Bit mit der Kapazität sinken und die Zugriffszeit mit der Kapazität steigt.
- Busse dienen dem Datentransfer zwischen Rechnerkomponenten.

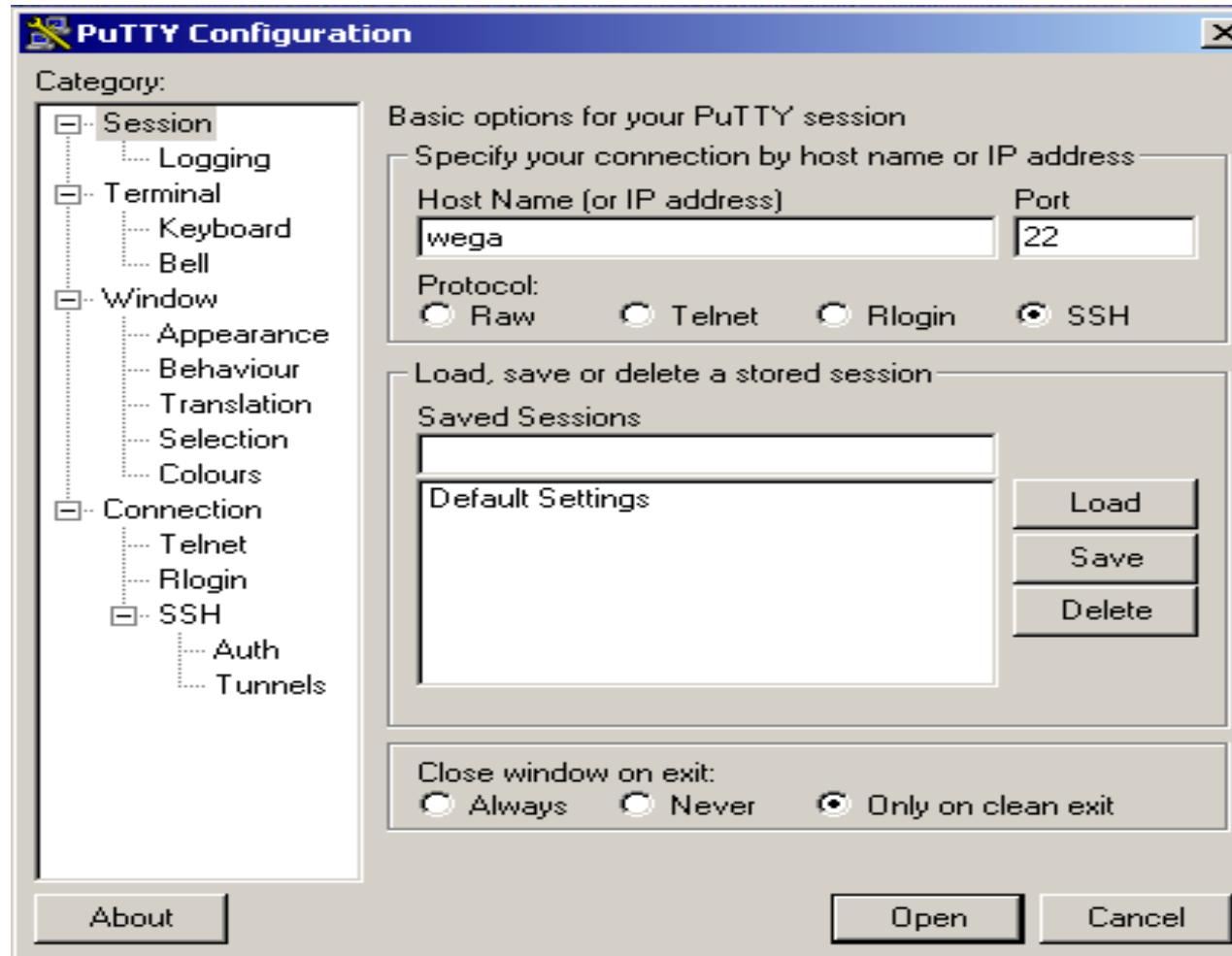
# **Kapitel 2 - Erste Schritte auf einem UNIX-System**

---

- 2.1 Anmeldung am System**
- 2.2 Informationsquellen unter UNIX**
- 2.3 Benutzeroauthentifizierung**
- 2.4 Der vi-Editor**

## 2.1 Anmeldung am System

- graphisch oder zeichenbasiert
- Zeichenbasiert: **putty** oder anderer Client



## 2.1 Anmeldung am System

---

- IP-Adresse: 143.93.203.25
- Port 22 (ssh)
- OpenSuse 13.2

## 2.1 Anmeldung am System

- „Sicherheitsabfrage vor dem erstmaligen Öffnen einer Verbindung:



- „Abfrage von Username und Passwort (Systemadministrator fragen)

## 2.2 Informationsquellen unter UNIX

---

typischer UNIX-Befehl:

```
$ command [-option [-option] ...] [parameter [parameter] ... ]
```

Erklärung in den Manualpages:

- UNIX-Reference Manual (ursprünglich)  
Volume 1 Section 1 user commands A-Z  
Volume 2 Section 2 system calls  
    Section 3 subroutine library

Volume 3 Section 1M                  System Administration  
    Section 4 File Formats  
    Section 5 Miscellaneous  
    Section 7 Device Files  
    Section 8 Glossary

```
$ man [chapter] [topic]  
$ man -k [topic]  
$ apropos [topic]
```

## 2.2 Informationsquellen unter UNIX

---

Aufbau einer Manalseite

Name, Kurzbeschreibung

Synopsis      Befehlssyntax

Description      ausführliche Beschreibung

Networking      Besonderheiten bezüglich Netzwerken

Features

Return Value      Rückgabewert des Kommandos

Diagnostics      Beschreibung der Fehlermeldung

Errors      Fehlerbedingungen

Warnings      s. Bugs

Hardware      ...

Dependencies

Author      urheberrechtlicher Eigner

Files      vom Kommando verwendete Dateien

International      Unterstützung von 8 Bit/16 Bit Zeichen

Support

Bugs      äußerst selten dokumentiert

See also      Querverweise zu anderen Befehlen

nicht alle Kapitel sind immer aufgeführt

## 2.2 Informationsquellen unter UNIX

---

### Aufgabe:

1. Welche Optionen besitzt der `man`-Befehl? Was bewirken diese?
2. Testen Sie diese an einem Beispiel!

## 2.3 Benutzeroauthentifizierung

---

- Userverwaltung in Datei /etc/passwd (für jeden Benutzer eine Zeile)

```
stud01:<verschlüsseltes PW>:<UID>:<GID>:Student, ABC, 721:/home/stud01:/bin/bash
```

Felder (durch Doppelpunkt getrennt) pro Zeile:

Login Name

Passwort

UID

GID

Beschreibung

Home-Verzeichnis

Login Shell (Kommandointerpreter)

aus Sicherheitsgründen oftmals Passwörter in /etc/shadow (nur von root lesbar)

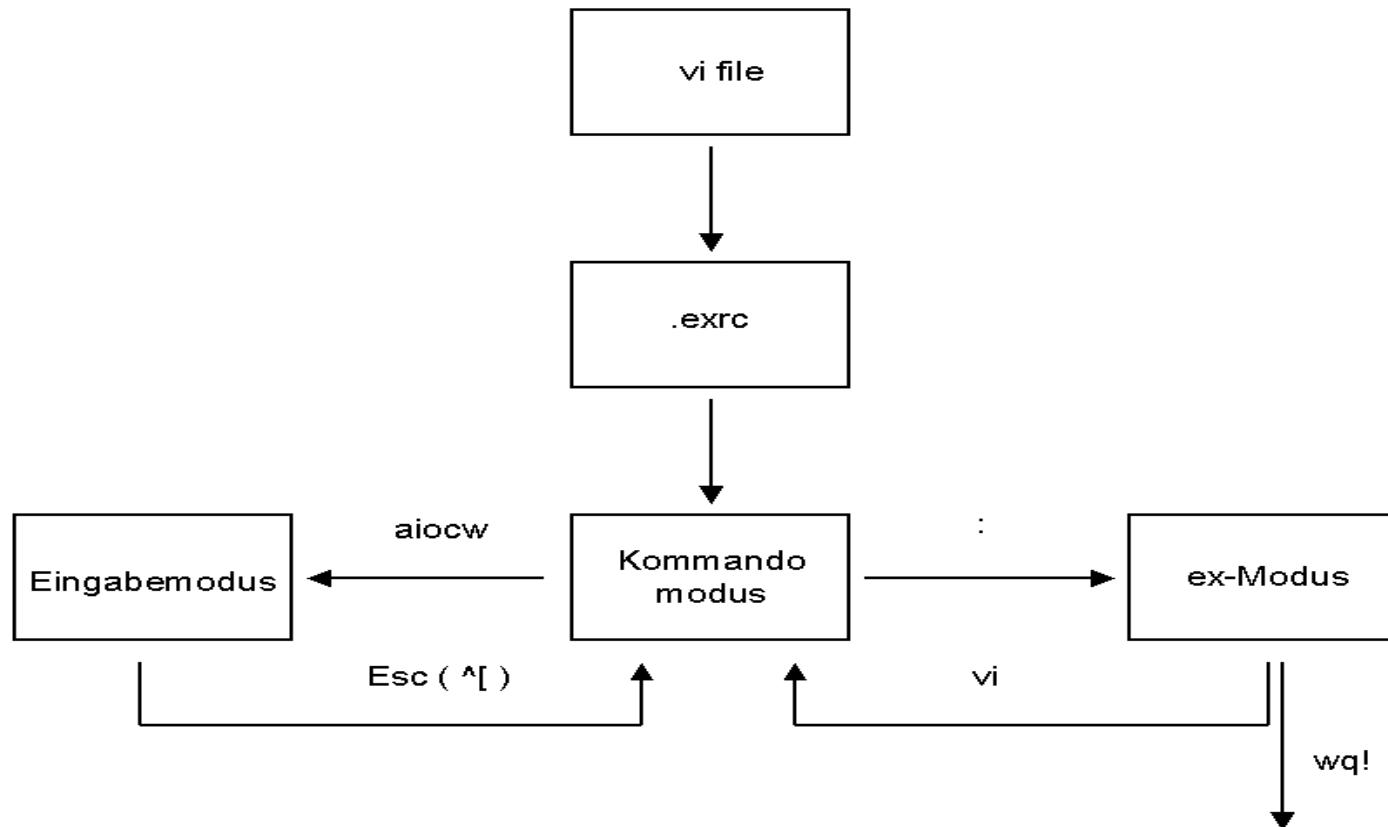
Passworteingabe beim Anmelden wird nicht ge-echo-t

### Aufgaben:

1. Finden Sie die Manualpage zur Datei /etc/passwd.
2. Schauen Sie sich die Manualpage an.
3. Welche Dateien sind relevant für die Passwortänderung?
4. Ändern Sie Ihr Passwort.
5. Melden Sie sich vom System ab und erneut wieder an.

## 2.4 Der vi-Editor

- auf jedem System verfügbar
- nicht graphisch (keine Mausunterstützung)
- keine Funktionstasten
- Option -r für Recovery
- Betriebsarten:



## 2.4 Der vi-Editor

---

Kommandoformat:

[count] command [where]

### Einige Kommandos zum Beginn

Kommando	Erläuterung
a	Übergang in den Eingabemodus. Neue Zeichen werden nach der aktuellen Cursorposition eingefügt.
i	Übergang in den Eingabemodus. Neue Zeichen werden vor der aktuellen Cursorposition eingefügt.
h/j/k/l	Bewegen des Cursors um ein Zeichen nach links/unten/oben/rechts. Diese Tasten sind sehr gut einsetzbar, wenn die Pfeiltasten auf der Tastatur vom vi nicht verstanden werden. Mitunter passiert dies, wenn die Netzwerkverbindung zum System, auf dem Sie arbeiten, sehr langsam ist und Stockungen in der Datenübertragung auftreten.
r	Ersetzen des Zeichens unter dem Cursor durch ein anderes
x	Löschen des Zeichens unter dem Cursor

## 2.4 Der vi-Editor

---

### Ausschneiden und Einfügen/Löschen von Text

Kommando	Erläuterung
<b>D</b>	Löschen bis zum Ende der Zeile
<b>P/p</b>	Einfügen vor/nach der aktuellen Cursorposition
<b>dd</b>	Löschen der aktuellen Zeile
<b>x</b>	Löschen des aktuellen Zeichens
<b>yy</b>	Kopieren der aktuellen Zeile

### Einfügen von neuem Text

Kommando	Erläuterung
<b>A</b>	am Ende der aktuellen Zeile einfügen
<b>I</b>	vom Zeilenanfang an einfügen
<b>O/o</b>	neue Zeile über/unter der aktuellen Cursorposition einfügen
<b>a/i</b>	Einfügen nach/vor der aktuellen Cursorposition

## 2.4 Der vi-Editor

---

### Cursorbewegungen

Kommando	Erläuterung
^B/^F	die nächste Seite nach oben/unten anzeigen
\$	den Cursor ans Zeilenende bewegen
%	finde die entsprechende öffnende oder schließende Klammer
G	gehe zur Zeile, welche als Count angegeben wurde. Falls kein Count angegeben wurde, gehe zum Ende des Files.
h/j/k/l	den Cursor ein Zeichen nach links/unten/oben/rechts bewegen
w	den Cursor zum Anfang des nächsten Wortes bewegen

# 2.4 Der vi-Editor

---

## Textersetzungen

Kommando	Erläuterung
C	Ersetzen bis zum Zeilenende
R	Wechsel in den Überschreibemodus bis zum Betätigen der Esc-Taste
c/cc	Ersetzen bis zum nächsten ":"/Ändern der aktuellen Zeile.
r	Überschreiben des Zeichens an der aktuellen Cursorposition
s	ein Zeichen unter dem Cursor ersetzen und in den Einfügemodus gehen

## Suchoperationen

Kommando	Erläuterung
/	Suche nach dem folgenden String abwärts
?	Suche nach dem folgenden String aufwärts
n	Wiederholung der letzten /- oder ?-Operation
N	Wiederholung der letzten /- oder ?-Operation, aber in der entgegengesetzten Richtung

## 2.4 Der vi-Editor

---

### Zeichen- und Zeilenformatierung

Kommando	Erläuterung
<code>~</code>	Umwandeln des aktuellen Zeichens unter dem Cursor von Groß- in Kleinbuchstaben und umgekehrt
<code>J</code>	die aktuelle und die folgende Zeile zu einer Zeile zusammenfassen

### Sichern und Verlassen des vi-Editors

<code>Q</code>	Verlassen des <i>vi</i> und Übergang in den <i>ex</i> -Modus. Zurück kommt man mit dem Kommando :vi.
<code>ZZ</code>	komplettes Verlassen des Editors und Sichern aller Änderungen

## 2.4 Der vi-Editor

---

### Sonstiges

<b>^G</b>	Anzeigen des aktuellen Dateinamens und dessen Status
<b>^L</b>	Leeren und Neuaufbau des Bildschirms
<b>^[</b>	alternative Esc-Taste, Abbrechen unvollständiger Kommandos
<b>^^</b>	zum zuletzt bearbeiteten File zurückgehen
<b>!</b>	Ausführen eines Shellkommandos
<b>. (Punkt)</b>	Wiederholen des letzten Editierkommandos
<b>:</b>	Übergang in den <i>ex</i> -Modus
<b>u</b>	das letzte Kommando rückgängig machen

## 2.4 Der vi-Editor

Der ex-Editor

Kommandos werden immer mit einem „:“ eingeleitet.

Kommando	Erläuterung
<code>:map keys sequence</code>	Die Tastensequenz <i>sequence</i> wird auf die Tasten <i>keys</i> gemappt.
<code>:q</code>	Verlassen der Datei ohne Speichern der Änderungen. Der Editor warnt, falls Veränderungen durchgeführt wurden.
<code>:q!</code>	Verlassen der Datei ohne Änderungen und ohne Nachfrage
<code>:s/from_pattern/to_pattern/g</code>	Ersetzen des Musters <i>from_pattern</i> durch <i>to_pattern</i> . Ohne Zusatz wird nur das erste Vorkommnis ersetzt. Das g ersetzt alle Vorkommnisse.
<code>:set [all]</code>	Setzen von Personalisierungsoptionen. Mit dem Zusatz all werden alle gegenwärtig aktiven Optionen angezeigt.
<code>:vi file</code>	Bearbeitung eines neuen Files im vi
<code>:w file</code>	Speichern der gemachten Änderungen in der Datei <i>file</i> .
<code>:w &gt;&gt;file</code>	Speichern der gemachten Änderungen durch Anhängen an die Datei <i>file</i> .
<code>:wq</code>	Speichern und Verlassen des Editors.

# Zusammenfassung Kapitel 2

---

- Die Anmeldung an einem UNIX/Linux-System sollte nach Möglichkeit über eine verschlüsselte Verbindung erfolgen, z.B. mit einem ssh-Client (ssh steht für Secure Shell). telnet und rlogin bieten keine Verschlüsselung, d.h. Passwörter und alle Daten werden unverschlüsselt in Klartext übertragen.
- Man kann sich über textbasierte oder grafische Clients an einem UNIX/Linux-System anmelden.
- Informationsquellen unter UNIX/Linux sind die Manual Pages, die mit dem Kommando man aufgerufen werden können.
- Die Benutzerverwaltung erfolgt in der Datei /etc/passwd. Passwörter werden eventuell in der Datei /etc/shadow gespeichert, die nur für root, d.h. den Systemadministrator lesbar ist.
- Mit dem Kommando passwd kann das Passwort geändert werden. Dies sollte auf jeden Fall nach dem ersten Login gemacht werden.
- Der vi ist ein Texteditor, der zeichenbasiert ist. Er bietet keine Mausunterstützung und wird komplett über Tasturbefehle bedient. Er ist auf jedem UNIX/Linux-System vorhanden und sollte daher in Grundzügen beherrscht werden.
- Die wichtigsten Kommandos sind i (insert), a (append), x (ein Zeichen löschen), dd (eine Zeile löschen), :wq (speichern und verlassen). Mit i und a wird in den Eingabemodus gewechselt. Mit der Esc-Taste wird der Modus wieder verlassen. Im Zweifelsfall sollte man vor der nächsten Aktion einmal die Esc-Taste drücken.

# Kapitel 3 - UNIX-Shells: Grundlagen

---

- 3.1 Eigenschaften der Shells**
- 3.2 Personalisierung der Arbeitsumgebung**
- 3.3 Subshells und Shellvariablen**
- 3.4 Dateinamenexpandierung**
- 3.5 Kommandosubstitution**
- 3.6 Ein-/Ausgabeumlenkung**
- 3.7 Quoting**
- 3.8 Pipelining**

# 3.1 Eigenschaften der Shells

---

- Shell ist Schnittstelle zwischen Anwender und Betriebssystem
- Kommandointerpreter
  - Aliase
  - Dateinamenergänzung
  - Kommandohistorie
  - Jobkontrolle
- Shell als Programmierumgebung: Befehlsflusskontrolle, flexible Ein-/Ausgabe, Funktionen, ...

## Vergleich der wichtigsten Shells

Shell	geeignet für interaktives Arbeiten	geeignet zur Skripterstellung	Syntax-Stil	Verfügbarkeit
<i>sh</i>	nein	Ja	<i>sh</i>	Standard
<i>ksh</i>	ja	Ja	<i>sh</i>	sehr gut
POSIX	ja	Ja	<i>sh</i>	gut
<i>bash</i>	ja	Ja	<i>sh</i>	gut
<i>csh</i>	ja	Nein	<i>csh</i>	Standard
<i>tcsh</i>	ja	Nein	<i>csh</i>	selten
<i>perl</i>	nein	Ja	<i>perl</i>	gut

## 3.2 Personalisierung der Arbeitsumgebung

---

- UNIX arbeitet mit Setup-Dateien
- meistens zwei Ebenen: systemweit (/etc), userbezogen(Punktdateien)

für die Kornshell gilt:

wenn Aufruf als Loginshell, dann Abarbeitung von:

- /etc/profile: für alle User
- dann ~/.profile
- Datei in \$ENV wird immer abgearbeitet

für die BASH gilt:

wenn Aufruf als Loginshell, dann Abarbeitung von:

- /etc/profile: für alle User
- dann ~/.bash\_profile, ~/.bash\_login, ~/.profile (erstes existierendes)
- Datei in \$BASH\_ENV wird immer abgearbeitet

## 3.2 Personalisierung der Arbeitsumgebung

---

- Variablen dienen der temporären Aufnahme von Informationen.
- Systemvariablen sind der Shell per Definition bekannt.
  - änderbare (in Setupdateien definierbar) und nicht änderbare (für Shell-Skripte)
  - Anzeige mit set bzw. env
- Variablennamen: Buchstaben, Ziffern (nicht am Anfang), Unterstrich (case sensitiv)
- Zugriff auf den Variableninhalt: vorangestelltes Dollarsymbol

### Beispiel: Einrichtung einer Arbeitsumgebung

~/.profile:

PATH=\$PATH:~/bin:..	:export PATH
PS1="`hostname` ! \$"	:export PS1
BASH_ENV=\$HOME/.bash_profile	:export BASH_ENV
EDITOR=vi	:export EDITOR

## 3.2 Personalisierung der Arbeitsumgebung

---

### Befehlsaliase

Abkürzung von häufig gebrauchten Kommandos

```
$ alias aliasname=command_string
```

Bsp.:

```
$ alias ll='ls -l'  
$ alias rm='rm -i'
```

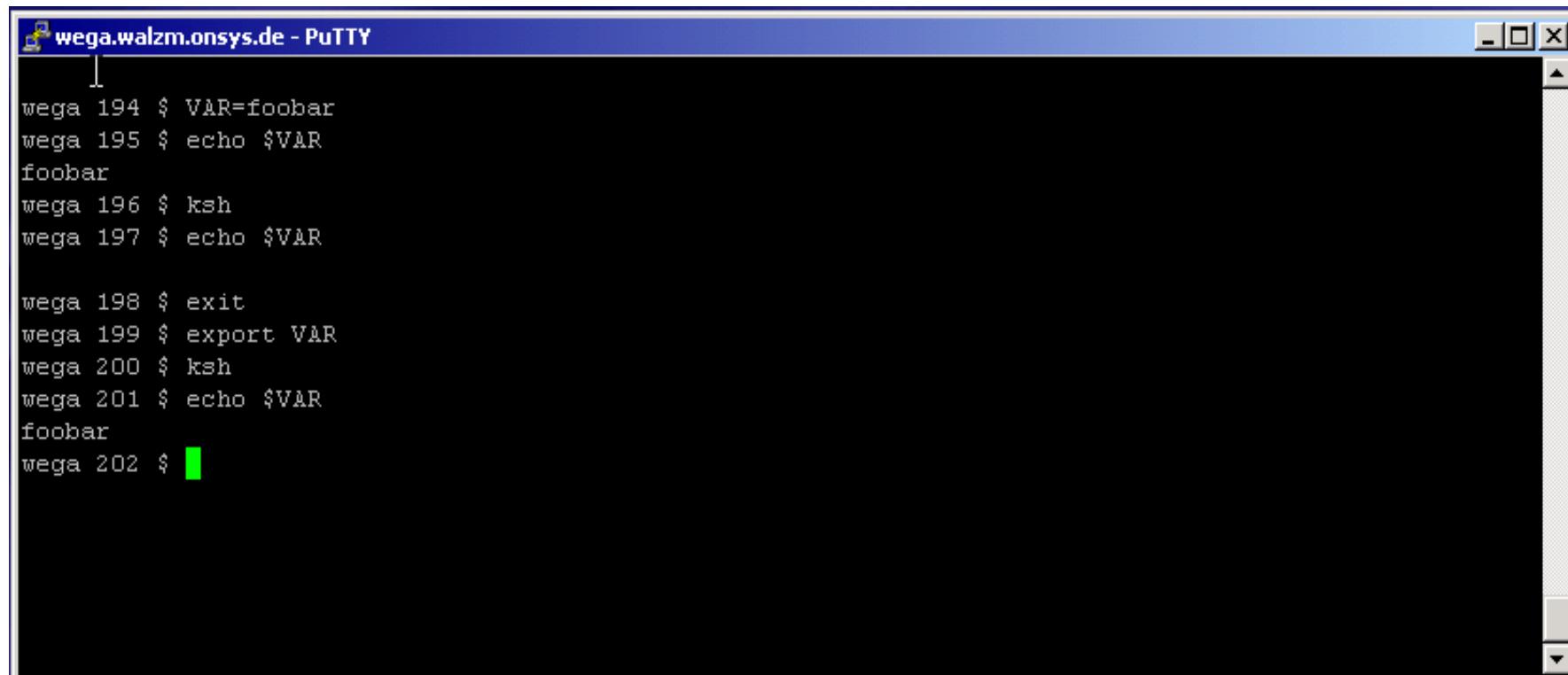
Übung:

1. Geben Sie die obigen Aliase auf der Kommandozeile ein und testen Sie die Aliase!
2. Testen Sie Befehlwiederholung (Cursortasten) und automatische Dateinamenergänzung (Tab-Taste) der Shell!

### 3.3 Subshells und Shellvariablen

- lokale (in der aktuellen Shell bekannt) und globale (Umgebungs-) Variablen
- durch Variablenexport werden lokale Variablen zu Umgebungsvariablen und sind damit in jeder Subshell bekannt
- Umgebungsvariablen können mit `env` angezeigt werden

```
$ VAR="wert"      # Definition der Variablen
$ echo $VAR       # Anzeige des Inhalts
$ export VAR      # zur Umgebungsvariable machen
```



The screenshot shows a PuTTY terminal window titled "wega.walzm.onsys.de - PuTTY". The terminal window has a blue header bar with the title and standard window controls (minimize, maximize, close). The main area is a black terminal window showing the following session:

```
wega 194 $ VAR=foobar
wega 195 $ echo $VAR
foobar
wega 196 $ ksh
wega 197 $ echo $VAR

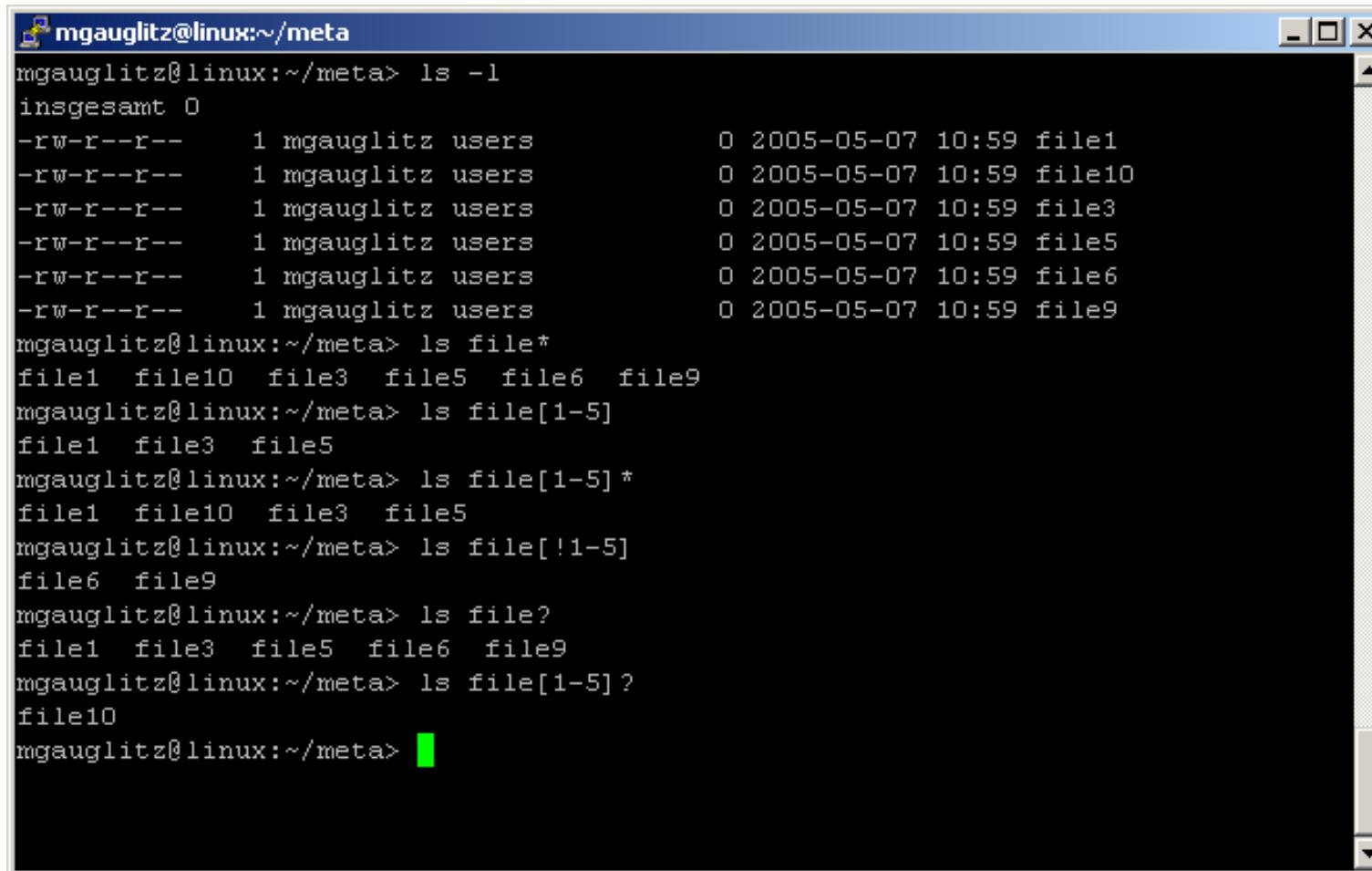
wega 198 $ exit
wega 199 $ export VAR
wega 200 $ ksh
wega 201 $ echo $VAR
foobar
wega 202 $
```

The terminal window has a vertical scroll bar on the right side.

# 3.4 Dateinamenexpandierung

## Metazeichen und ihre Bedeutung

- \* Eine beliebige Zeichenfolge (auch leer)
- [ ... ] Eines der in [ ... ] angegebenen Zeichen
- [! ... ] Ein Zeichen, welches nicht in [ ... ] angegeben ist
- ? Ein beliebiges einzelnes Zeichen



The screenshot shows a terminal window titled "mgauglitz@linux:~/meta". It displays several commands demonstrating file expansion:

- ls -l: Shows a list of files: file1, file10, file3, file5, file6, and file9.
- ls file\*: Shows the same list of files: file1, file10, file3, file5, file6, and file9.
- ls file[1-5]: Shows files file1, file3, and file5.
- ls file[1-5] \*: Shows files file1, file10, file3, and file5.
- ls file[!1-5]: Shows files file6 and file9.
- ls file?: Shows files file1, file3, file5, file6, and file9.
- ls file[1-5] ?: Shows file10.

## 3.5 Kommandosubstitution

---

Die Ausgabe eines Kommandos kann als Teil der Kommandozeile selbst interpretiert werden.

### Zwei Formen

- Einschluss des Kommandos durch \$( . . )
- Klammerung mit Gegenapostrophen (Backquotes) ` . . `

### Beispiel:

```
$ echo "Heute ist der $(date +%d.%m.%Y)."  
$ echo "Heute ist der `date +%d.%m.%Y`."
```

## 3.6 Ein-/Ausgabeumlenkung

UNIX ordnet jedem Prozess per Default drei E/A-Kanäle zu.

- &0: stdin (Standardeingabe), Tastatur
- &1: stdout (Standardausgabe), Bildschirm
- &2: stderr (Standardfehlerausgabe), Bildschirm

Die E/A – Ströme können umgelenkt werden.

Zweck	Kurzform	ausführlich	Beispiel
Ausgabe in File (stdout)	>file	1>file	date >Datum
Ausgabe an File anhängen (stdout)	>>file	1>>file	date >>Datum
Fehlerausgabe in File (stderr)	2>file		2>&1
Standardeingabe aus File lesen	<file	0<file	cat <datei
Standardeingabe aus einem „here-Dokument“ lesen	<<WORD	0<<WORD	⇒ siehe Kapitel Shell-Programmierung

## 3.6 Ein-/Ausgabeumlenkung

---

Beispiele:

```
$ cmd 1>/dev/null 2>&1
```

=> lenkt stdout nach /dev/null und stderr nach stdout und somit auch nach /dev/null

```
$ cmd >/tmp/test.log 2>&1
```

=> lenkt stdout in Datei /tmp/test.log und stderr nach stdout und somit auch nach /tmp/test.log

# 3.7 Quoting

---

Quoting heisst: die Bedeutung der Sonderzeichen in der Shell auszuschalten.

Sonderzeichen: \$, " , ', ` , \, ?, \*

Drei Varianten:

- Maskieren mit dem Backslash (\):

- Das nachfolgende Sonderzeichen verliert seine Bedeutung.
- Innerhalb ' .. ' gilt \ nicht als Sonderzeichen.
- Innerhalb " .. " schaltet \ nur die Bedeutung von \$, ` , \, " aus.

- Klammerung mit ' .. ':

- Alle Metazeichen, ausser ' verlieren ihre Bedeutung.

Beispiel:

```
$ VAR="Das ist der Inhalt der Variablen VAR."
$ echo '$VAR'
$VAR
```

- Klammerung mit " .. ":

- \, ", ' , \$ behalten ihre Bedeutung.

Beispiel:

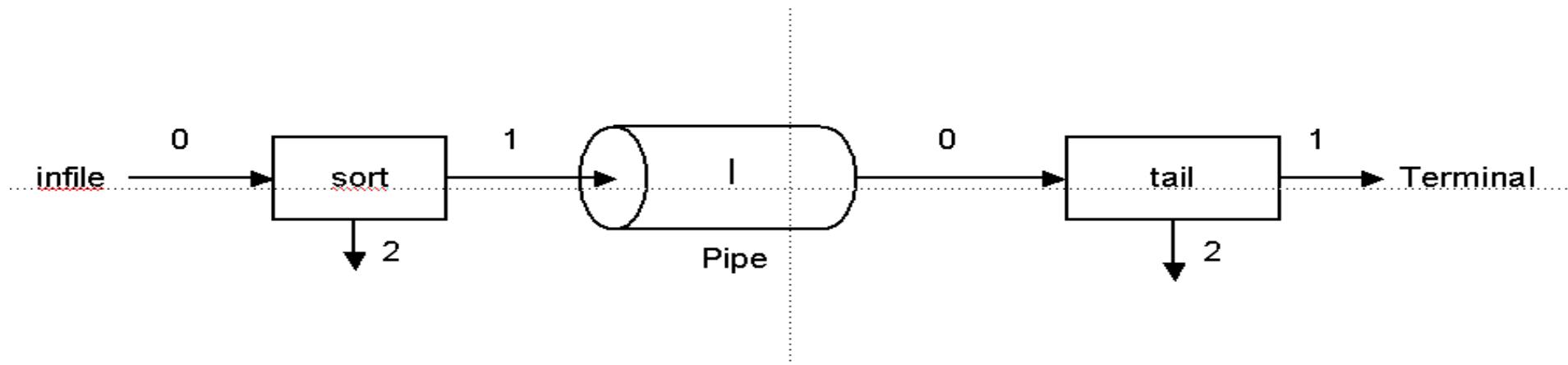
```
$ VAR="Das ist der Inhalt der Variablen VAR."
$ echo "$VAR"
Das ist der Inhalt der Variablen VAR.
```

# 3.8 Pipelining

- Pipelining ist eines der mächtigsten UNIX-Konzepte.
- Verknüpfung der Standardausgabe eines Programms mit der Standardeingabe eines anderen Programms.
- Aufbau von Befehlsketten möglich.
- Pipe: Pseudodatei zur Verbindung von Dateideskriptoren und zur Datenpufferung.

Beispiel:

```
$ sort <infile | tail -10
```



# Zusammenfassung Kapitel 3

---

- Die Shell ist eine zeilenbasierte Schnittstelle zwischen Anwender und Betriebssystem.
- Die Shell ist ein Kommandointerpreter und eine Programmierumgebung für Shellskripte.
- Über Setupdateien ist die Arbeitsumgebung in der Shell personalisierbar.
- In der Shell gibt es lokale Variablen und Umgebungsvariablen. Per Export können lokale zu Umgebungsvariablen gemacht werden.
- Die Shell bietet verschiedene Features wie Dateinamenergänzung, Kommandosubstitution, Ein- und Ausgabeumlenkung und Pipelining.
- Bei der Dateinamenergänzung gibt es Metazeichen, die für Teile eines Dateinamens stehen können.
- Bei der Kommandosubstitution wird statt eines Kommandos das Ergebnis des Kommandos verarbeitet.
- Die Standardeingabe, die Standardausgabe und die Standardfehlerausgabe eines Kommandos können in Dateien umgelenkt werden.
- Durch Pipelining können Befehlsketten gebildet werden. Die Ausgabe eines Kommandos wird dabei als Eingabe des nächsten Kommandos verwendet. Die Verkettung erfolgt über das | -Zeichen (Pipe-Zeichen).

# **Kapitel 4 – BS-Konzepte: Prozessmanagement**

---

**4.1 Prozesse**

**4.2 Threads**

**4.3 Scheduling**

**4.4 Prozessmanagement unter UNIX/Linux**

# 4.1 Prozesse

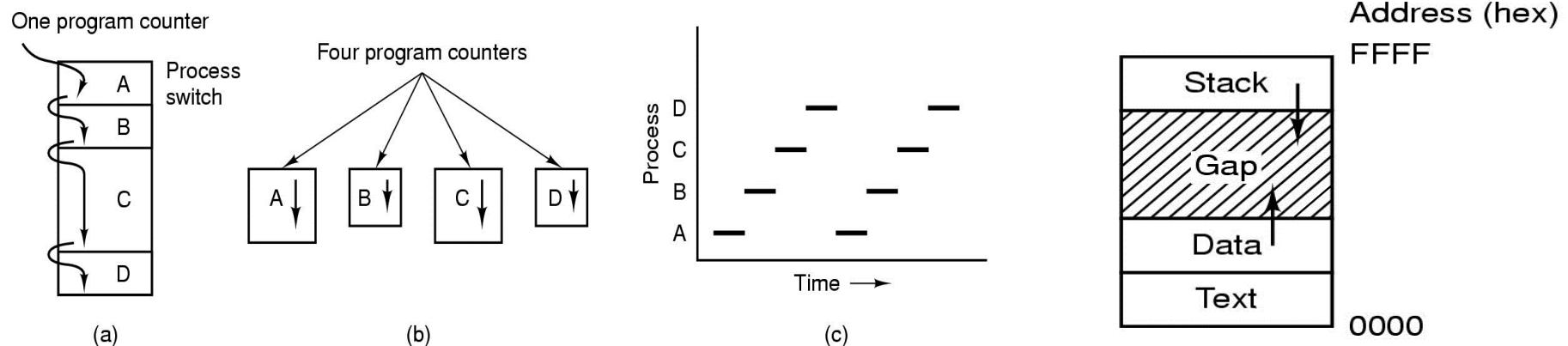
## Was ist ein Prozess?

Ein Prozess ist ein Programm in Ausführung inklusive der aktuellen Werte seines Befehlszählers, der Registerinhalte, der Variablenbelegungen, des Stack, des Zustands der geöffneten Dateien und Netzwerkverbindungen, der Programmdaten und der Speicherinformationen.

Zu einem bestimmten Zeitpunkt kann nur eine Aufgabe bearbeitet werden, CPU schaltet im ms-Bereich -> Illusion der Parallelität (Quasiparallelität, Multiprogramming).

### Prozessmodell

- Die gesamte ausführbare Software inkl. Betriebssystem ist als eine Menge von sequentiellen Prozessen organisiert.
- Konzeptionell besitzt jeder Prozess eine eigene virtuelle CPU (b), in der Praxis nur eine CPU, die zwischen Prozessen umschaltet ((a) und (c))
- Ein Prozess darf keine Annahmen über den absoluten Zeitablauf machen.
- Speicherinhalt jedes Prozesses besteht aus drei Segmenten: Text, Data, Stack.



# 4.1 Prozesse

---

## Erzeugen und Beenden von Prozessen

**Es gibt vier Möglichkeiten, einen Prozess zu erzeugen:**

- 1. Beim Systemstart (Initialisierung)**
- 2. Durch einen anderen Prozess (Systemaufruf z.B. *fork*)**
- 3. Interaktiv durch den Benutzer**
- 4. Per Stapelverarbeitung (Batch Job)**

**Analog dazu können Prozesse beendet werden durch:**

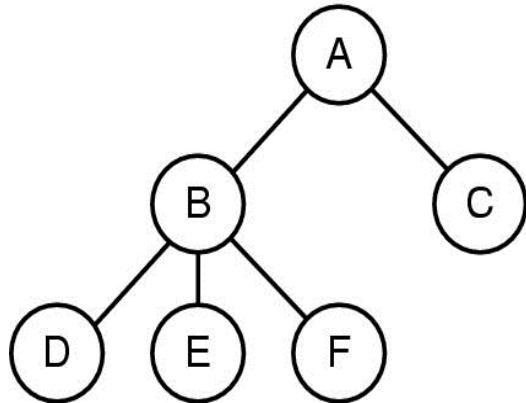
- 1. Normales Beenden (freiwillig)**
- 2. Aufgrund eines Fehlers (freiwillig)**
- 3. Aufgrund eines „schweren“ Fehlers (unfreiwillig)**
- 4. Durch einen anderen Prozess (unfreiwillig)**

# 4.1 Prozesse

---

## Prozesshierarchien

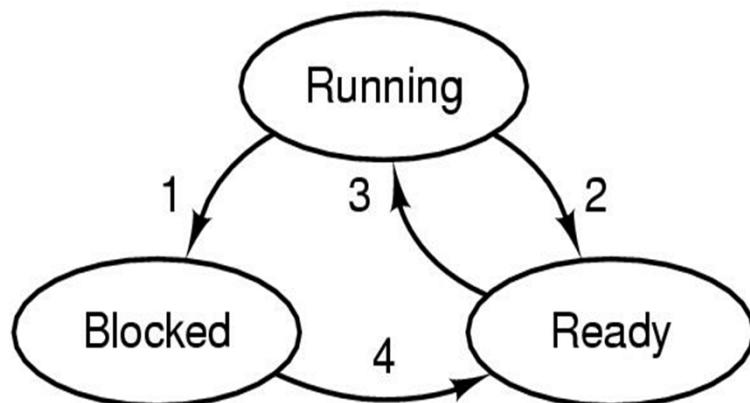
- Vaterprozess erzeugt Kindprozess, Parent erzeugt Child, diese wiederum weitere Childs.
- Bsp.: Unix: im Bootimage liegt spezieller Prozess: init
  - dies startet Hintergrundprozesse und Prozess pro Terminal
  - warten auf Login
  - Ausführen von Loginshells
  - Kommandoeingaben führen zu weiteren Prozessen



# 4.1 Prozesse

## Prozesszustände

- Multiplexen des physischen Prozessors übernehmen Scheduler und Dispatcher
- Prozesszustände bilden Entscheidungsgrundlage für die Auswahl eines geeigneten Kandidaten bei einem Prozesswechsel durch Scheduler und Dispatcher



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- in der Praxis werden Prozesszustände feiner unterteilt:

Beispiel: Zustand blockiert kann in realem Betriebssystem auf *warten\_auf\_Zeitereignis* und *warten\_auf\_Ein/Ausgabe* abgebildet sein

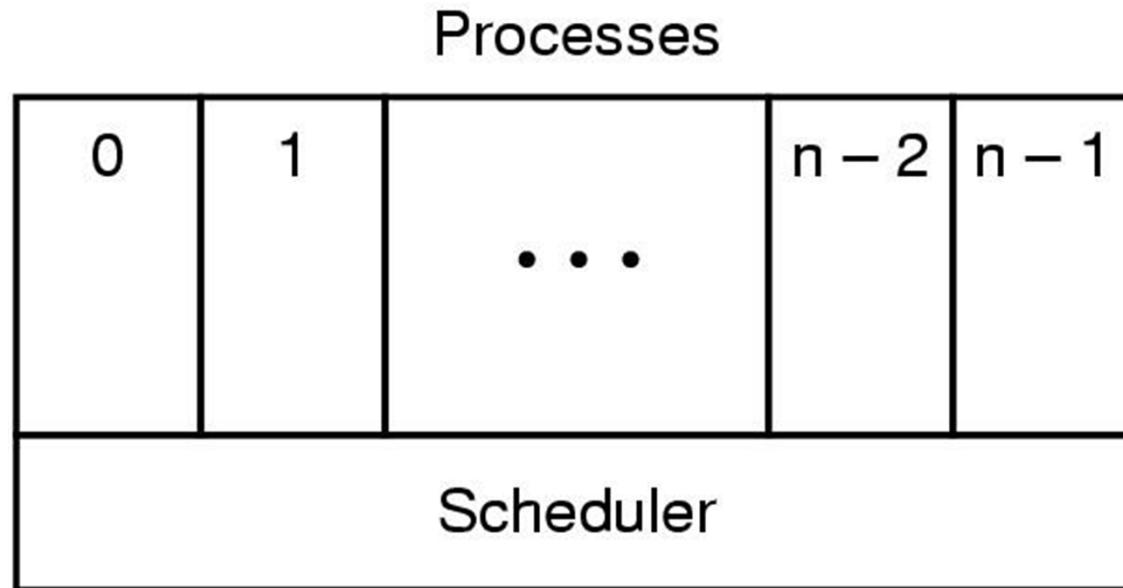
## 4.1 Prozesse

---

- Dispatcher: ist verantwortlich für die Zustandsübergänge
- Scheduler: Auswahl eines rechenbereiten Prozesses

prozessorientiertes Modell des Betriebssystems:

- Unterste Schicht: Scheduler und Dispatcher (Interruptbehandlung, Erzeugung und Abbrechen von Prozessen)
- alle anderen Prozesse auf einer Ebene und haben sequentiellen Kontrollfluss



# 4.1 Prozesse

## Prozesstabelle

Implementierung des beschriebenen Prozessmodells

Speicherung aller relevanten Informationen durch das Betriebssystem, um Prozesse zwischen den verschiedenen Zuständen schalten zu können:

<b>Process management</b>	<b>Memory management</b>	<b>File management</b>
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

## 4.2 Threads

---

### Threads

- ein Thread ist ein Ausführungsfaden
- traditionelles Betriebssystem: ein Prozess hat einen Thread und einen eigenen Adressraum
- Mehrere Threads können sich den selben Adressraum teilen => Multithreading
- Threads sind leichtgewichtige Prozesse (Threadwechsel schneller als Prozesswechsel)

#### **Per process items**

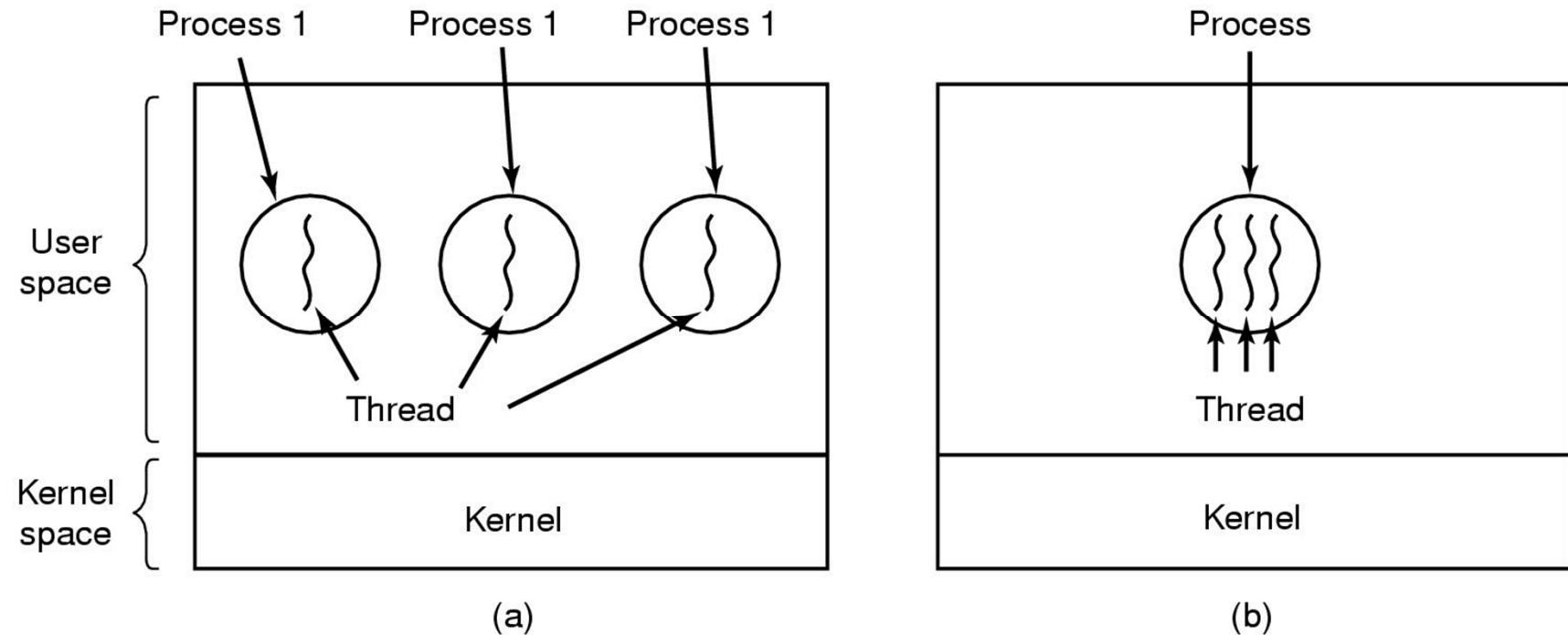
Address space  
Global variables  
Open files  
Child processes  
Pending alarms  
Signals and signal handlers  
Accounting information

#### **Per thread items**

Program counter  
Registers  
Stack  
State

## 4.2 Threads

Multithreading: schnelles Hin- und Herschalten zwischen Threads, die sich den gleichen Adressraum teilen, also innerhalb eines Prozesses liegen, zur Erzeugung der Illusion der Parallelität, gleiche Zustände wie bei Prozessmodell



- (a) Drei Prozesse mit je einem Thread
- (b) Ein Prozess mit drei Threads

# 4.2 Threads

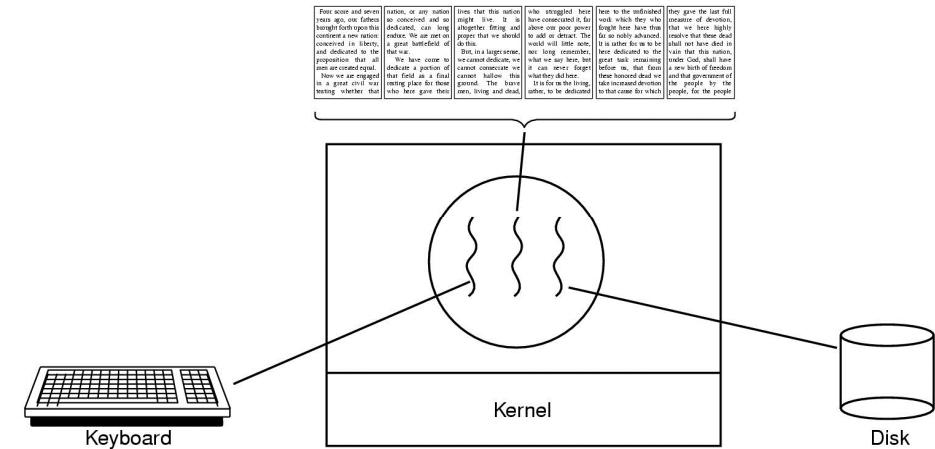
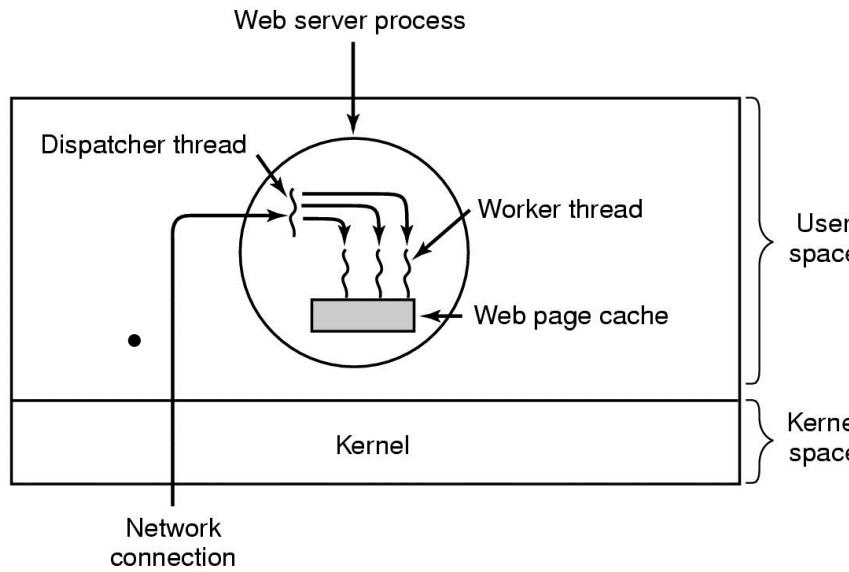
## Vorteile von Threads

- Einfachheit: mehrere Aktivitäten in Anwendungen gleichzeitig, Blockierung einzelner Aktivitäten => Zerlegung in mehrere quasiparallele Threads, so dass nicht gesamter Prozess blockiert, sondern nur ein Thread
- Keine Ressourcenbindung: leichter zu erstellen und zu zerstören als Prozesse
- Performance: effiziente Überlappung von I/O intensiven Aktivitäten (langsam) und CPU-intensiven Aktivitäten (schnell)
- auf SMP Systemen (mehrere CPUs): echte Parallelität

Bsp.:

Webserver

Textverarbeitung

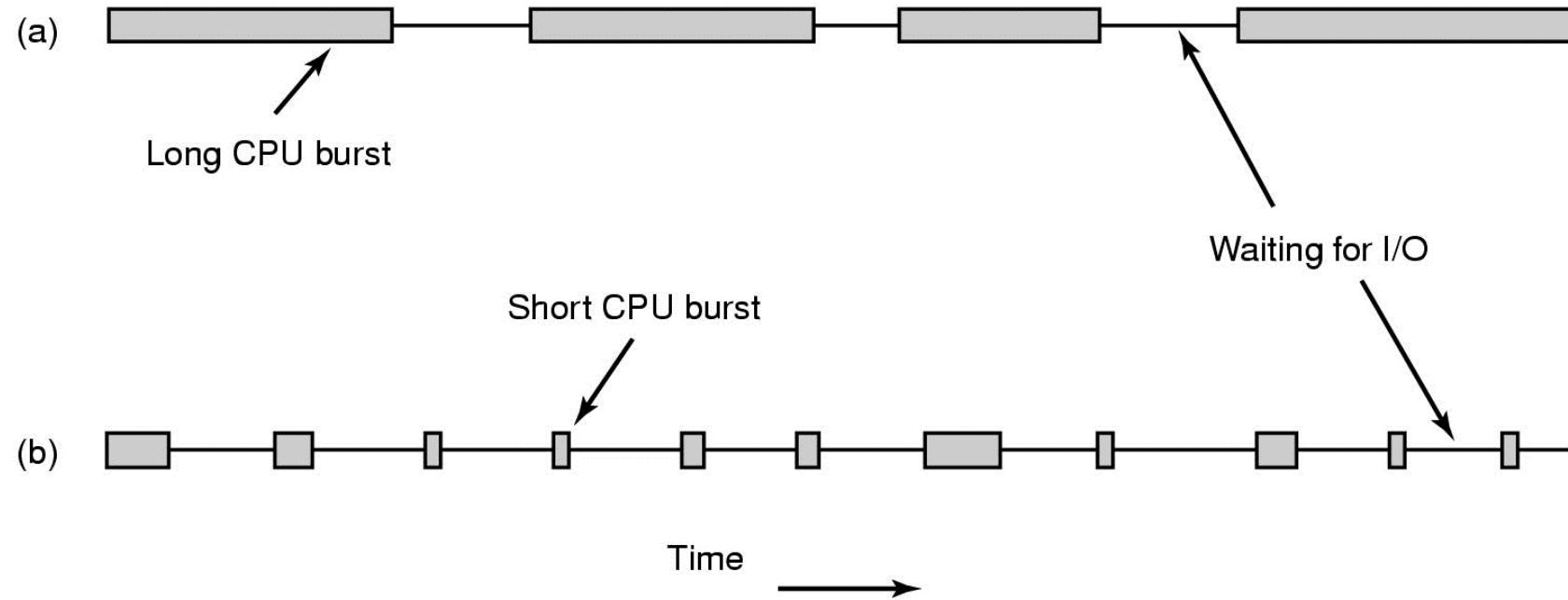


# 4.3 Scheduling

## Scheduling

Der Scheduler ist der Teil des Betriebssystems, welcher die Wahl trifft, welcher der bereiten Prozesse als nächstes Rechenzeit der CPU bekommt.

- Prozessarten: CPU lastig, I/O lastig, in der Praxis wechseln Prozesse zwischen beiden Arten



# 4.3 Scheduling

---

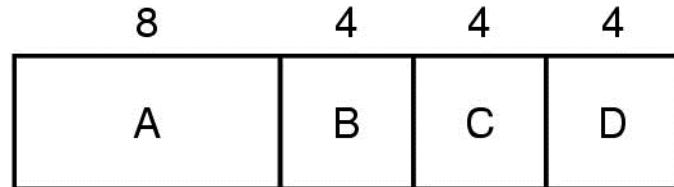
- Grundlegende Fragen:
  - wer rechnet bei Prozessentstehung (Child/Parent)
  - wer rechnet bei Prozessbeendigung
  - Auswahl eines anderen Prozesses, falls vorgesehener Prozess wegen I/O blockiert
  - was passiert bei I/O Interrupts
  - nicht unterbrechende (nicht präemptive) Scheduler:  
lässt Prozess solange laufen, bis er blockiert oder beendet ist
  - unterbrechende (präemptive) Scheduler:  
lässt Prozesse nur festgelegte Zeit (Quantum) laufen und unterbricht dann
- Kategorien von Scheduling Algorithmen:
  - Stapelverarbeitung
  - Interaktion
  - Echtzeit

# 4.3 Scheduling

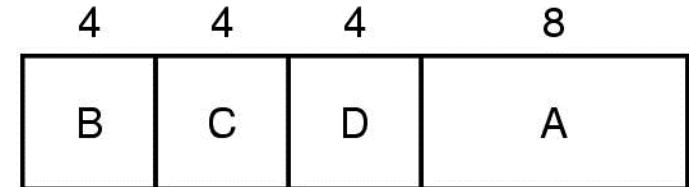
## Scheduling in Batch Systemen

Nicht präemptive Algorithmen:

- First-Come First-Served: gerecht, einfach zu implementieren, aber nicht effizient (a)
- Shortest Job First: optimal, aber nicht gerecht (b)



(a)



(b)

Beweisskizze, dass Shortest Job First optimal:

Annahme: vier Prozesse mit Laufzeiten a, b, c und d

Erste Aufgabe endet nach Zeit a

zweite nach a+b

dritte nach a+b+c

vierte nach a+b+c+d

⇒ Durchschnittliche Laufzeit ist  $(4a + 3b + 2c + d) / 4$

⇒ a trägt am meisten zum Durchschnitt bei, b am zweitmeisten, c am drittmeisten, d am wenigsten

Präemptiver Algorithmus:

- Shortest Remaining Time Next

# 4.3 Scheduling

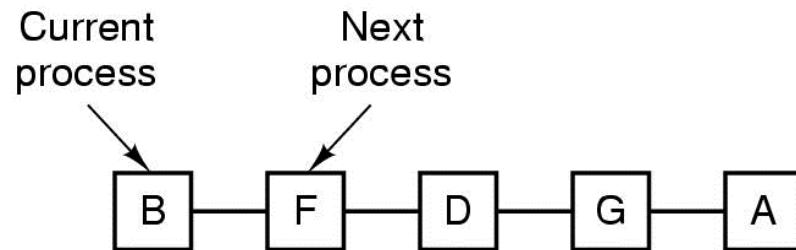
## Scheduling in interaktiven Systemen

Präemptive Algorithmen (auch in Batchsystemen einsetzbar):

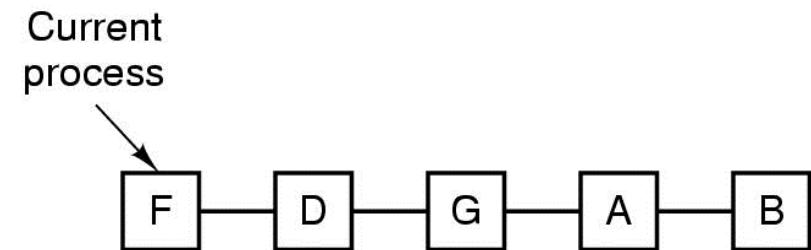
- Round – Robin – Scheduling:

Annahme: jeder Prozess gleich wichtig

jedem Prozess wird ein gleich grosser Zeitabschnitt (Quantum) zugewiesen



(a)



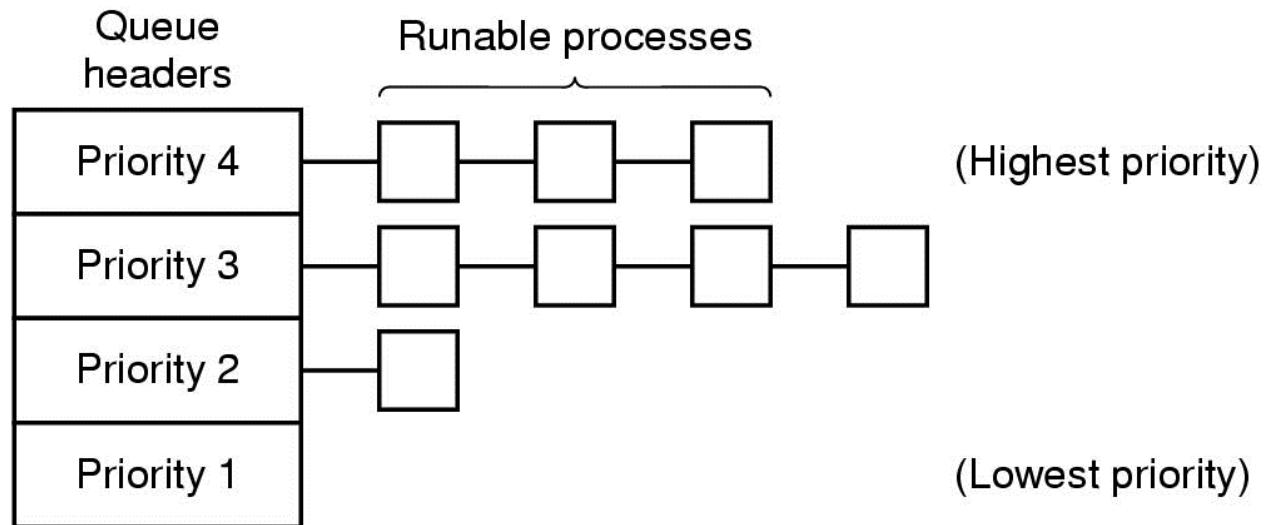
(b)

- Frage: Länge des Quantums
- Annahme Prozess- bzw. Kontextwechsel 1ms, Quantum 4ms  
-> 20% der CPU-Zeit Verwaltungsoverhead
- Verbesserung CPU-Effizienz: Quantum 100ms -> 1% Verwaltungsoverhead, aber bei zehn gleichzeitigen Eingaben 1s Wartezeit
- Kompromiss zwischen CPU-Effizienz und Antwortzeiten: 20 bis 50 ms Quantum

# 4.3 Scheduling

## Scheduling in interaktiven Systemen

- Priority – Based – Scheduling:
  - Prozesse bekommen Priorität, Prozess mit höchster Priorität darf laufen
  - statische oder dynamische Prioritäten
  - Zusammenfassung zu Prioritätenklassen, prioritätenbasiertes Scheduling zwischen Klassen
  - innerhalb der Klassen Round – Robin – Scheduling



- Lottery – Scheduling: Scheduler gibt für verschiedene Systemressourcen Lose aus
  - Bsp: CPU, Prozess mit Los bekommt 20ms Rechenzeit
  - => definierte Priorität (10 Lose für einen Prozess bedeuten 200ms Rechenzeit)
- Fair – Share – Scheduling: Rechenzeit wird abhängig gemacht vom Eigentümer der Prozesse
  - egal wieviele Prozesse er besitzt

# 4.4 Prozessmanagement unter UNIX/Linux

---

## Realisierung von Prozessen in UNIX

- Kernel verwaltet zwei Datenstrukturen Prozesstabelle und Benutzerstruktur

### Prozesstabelle

- resident, enthält Infos, die für alle Prozesse (auch ausgelagerte) wichtig sind
- Parameter für das Scheduling (Priorität, verbrauchte CPU-Zeit, kürzlich schlafende Zeit)
- Speicherabbild (Zeiger auf Text-, Daten- und Stacksegment bzw. ihre Seitentabellen)
- Signale (Masken mit ignorierte/aufgefangene/blockierte/zugestellte Signale)
- Verschiedenes (Zustand, blockierende Ereignisse, PID Vater, UID, GID)

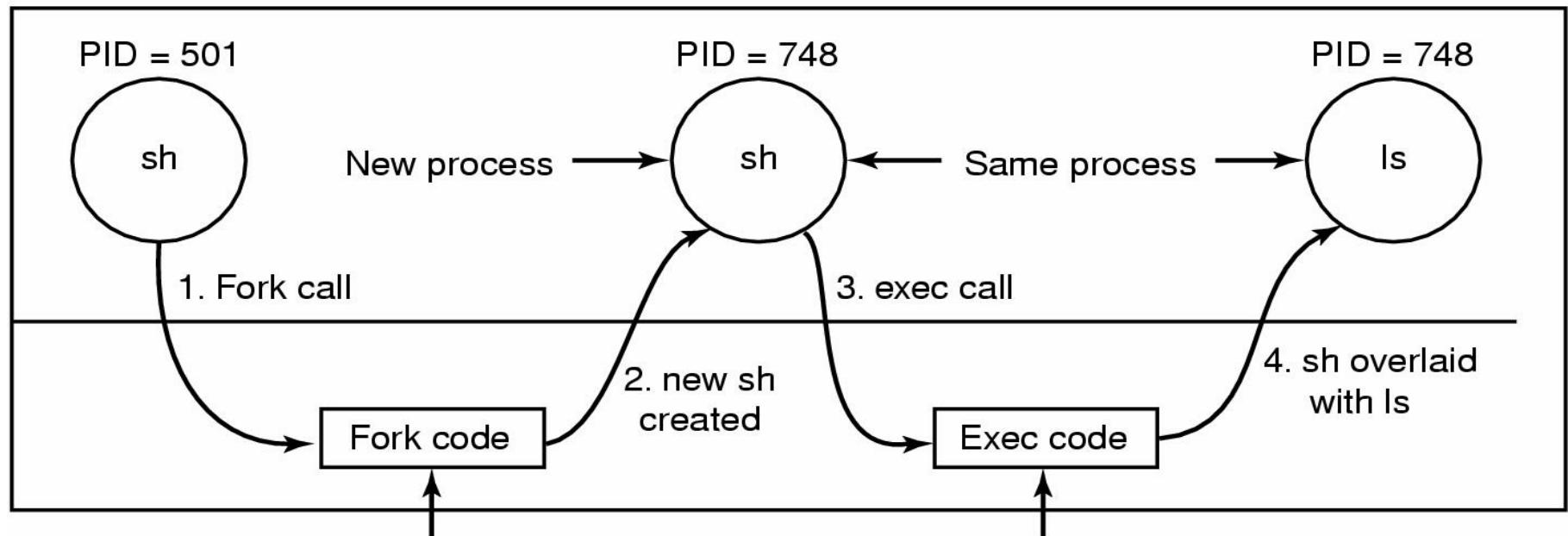
### Benutzerstruktur:

- enthält Infos, die nicht benötigt werden, wenn Prozess ausgelagert ist
- Maschinenregister (bei Sprung in Kernel Sicherung der Register)
- Zustand der Systemaufrufe (aktueller Aufruf inkl. Parameter und Ergebnis)
- Dateideskriptorentabelle (zum Finden des I-Node)
- Buchhaltung (z.B. vom Prozess verwendete Benutzer- und System-CPU-Zeit)
- Kernel-Stack (Stack für Kernelanteil des Prozesses)

# 4.4 Prozessmanagement unter UNIX/Linux

## Prozesserzeugung unter UNIX

Prozesse entstehen immer aus Prozessen. (Parent -> Child)



Allocate child's process table entry  
Fill child's entry from parent  
Allocate child's stack and user area  
Fill child's user area from parent  
Allocate PID for child  
Set up child to share parent's text  
Copy page tables for data and stack  
Set up sharing of open files  
Copy parent's registers to child

Find the executable program  
Verify the execute permission  
Read and verify the header  
Copy arguments, environ to kernel  
Free the old address space  
Allocate new address space  
Copy arguments, environ to stack  
Reset signals  
Initialize registers

## 4.4 Prozessmanagement unter UNIX/Linux

---

Zu jedem Prozess gehören drei Dateien:

- Standardausgabe (stdout: 1)
- Standardeingabe (stdin: 0)
- Standardfehlerausgabe: (stderr: 2)

Beim Start aus der Shell übernimmt das Terminal die Standardwerte der drei Kanäle:  
stdin von Tastatur, stdout/stderr auf Bildschirm

Jedem Prozess sind eine Userid (UID) und effektive UID zugeordnet:

UID: Identifikationsnummer des startenden Benutzers

EUID: bestimmt, auf welche Ressourcen der Prozess zugreifen darf

=> Unterscheidung zwischen Identität und Rechten

# 4.4 Prozessmanagement unter UNIX/Linux

---

## Hintergrundprozesse

```
$ cmd [-opt] [args] &
```

- der Parent-Prozess wartet **nicht** bis zur Beendigung des Child-Prozesses
- cmd wird im Hintergrund ausgeführt
- PID wird ausgegeben
- Shell-Prompt erscheint sofort wieder
- die Standard-Eingabe wird vom Terminal abgekoppelt (/dev/null).
- die Standard-Ausgabe bleibt unberührt.
- Um Prozesse auch nach Beenden des Vordergrundprozesses im Hintergrund weiterlaufen zu lassen, verwendet man das Kommando „nohup“ (Immunisierung gegen HUP-Signal)

```
$ nohup cmd [-opt] [args] &
```

Die Standard-Ausgabe wird nach nohup.out umgelenkt

# 4.4 Prozessmanagement unter UNIX/Linux

---

## Scheduling in UNIX

- optimiert für Antwortzeiten in interaktiven Systemen

CPU-Scheduler:

- mehrere Warteschlangen (Prioritäten) (siehe folgende Seite)
- innerhalb einer Warteschlange: Round Robin
- im Benutzermodus positive Werte, im Kernelmodus negative Werte
- nur Prozesse in Warteschlangen, die im Speicher und rechenbereit sind
- Prozess mit höchser Priorität bekommt für ein Quantum (typisch 100ms) Rechenzeit
- Prozess wird danach ans Ende der Warteschlange gestellt
- einmal pro Sekunde Neuberechnung der Priorität:

$$p = \text{cpu\_usage} + \text{nice} + \text{base}$$

cpu\_usage: Anzahl Uhrticks, die der Prozess die CPU benutzt hat

nice: oft -20 bis +20, durch Systemaufruf nice setzbar (nur 0 bis 20)

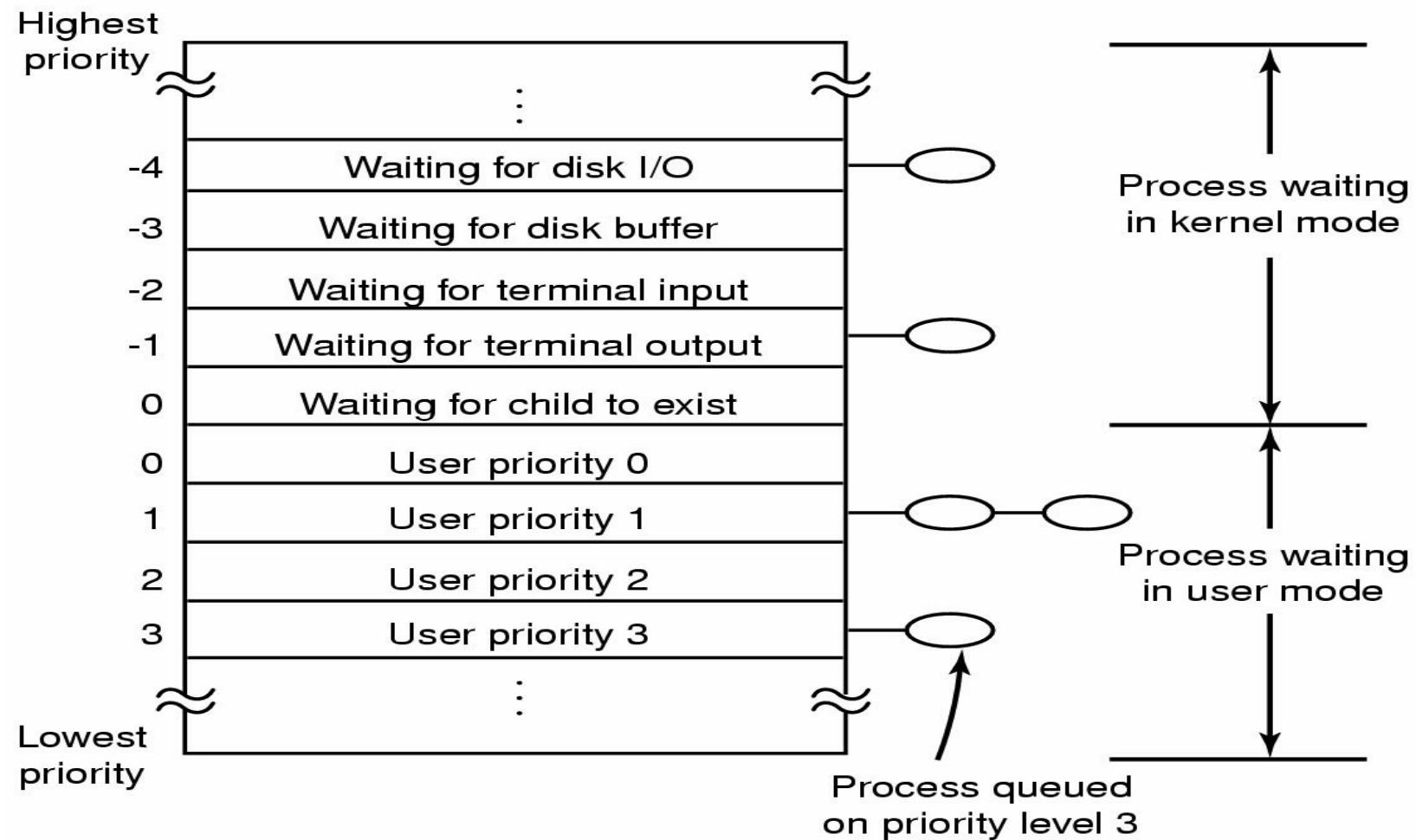
base: Einstufung nach Operation des Prozesses (fest im System)

=> führt eventuell zu einem Wechsel der Warteschlange

Ergebnis: interaktive Prozesse und Prozesse mit E/A (Kernelspace muss frei werden)  
werden bevorzugt

Linux benutzt anderen Scheduling-Algorithmus

# 4.4 Prozessmanagement unter UNIX/Linux



# 4.4 Prozessmanagement unter UNIX/Linux

---

## Prozessmonitoring mit ps

ps: Informationen über Prozesse, welche dem aktuellen Terminal zugeordnet sind.  
ps -aef: Informationen über alle momentan aktiven Prozesse

- UID: Benutzerkennung des Prozesses
- PID: Prozess-ID
- PPID: Parent-Prozess-ID
- C: Inanspruchnahme der CPU
- STIME: Startzeit des Prozesses
- TTY: Terminal, dem der Prozess zugeordnet ist
- TIME: bisherige Ausführungszeit
- CMD: der Befehl, der aktiv ist

## Übungen



Schauen Sie sich die Prozessliste des Systems an,  
suchen Sie Ihre Login-Shell und verfolgen Sie die Prozess-Hierarchie bis zur  
kleinstmöglichen Prozess-ID.

# 4.4 Prozessmanagement unter UNIX/Linux

---

## Prozessmonitoring mit top

Sortierung der aktiven Prozesse nach ihrer Aktivität

drei Arten von Informationen:

allg. Systeminformationen

aktuelle Zeit, Uptime, Belastungsmittelwerte des Prozessors

Anzahl Prozesse, Anzahl in einzelnen Prozesszuständen

CPU-Benutzung in den einzelnen Bereichen (user, system, nice, idle)

Speicherinformationen

Memory und Swap (total, used, free ...)

Prozessinformationen

PID: Prozess ID

USER: Owner des Prozesses

PR: aktuelle Priorität

NI: aktueller nice-Wert

VIRT: Grösse des Prozesses (SWAP + RES)

RES: Anteil des Prozesses im Hauptspeicher

SHR: Shared Memory

S: aktueller Zustand des Prozesses

%CPU: Anteil an der CPU-Last

%MEM: Anteil an physischem Speicher

TIME: Anzahl verbrauchter System- und CPU-Sekunden

COMMAND: Kommandoname des Prozesses

# 4.4 Prozessmanagement unter UNIX/Linux

```
mgauglitz@linux:~  
top - 20:08:51 up 44 days, 8:49, 1 user, load average: 0.22, 0.05, 0.02  
Tasks: 46 total, 1 running, 45 sleeping, 0 stopped, 0 zombie  
Cpu(s): 1.0% user, 1.0% system, 0.0% nice, 98.0% idle  
Mem: 126284k total, 123652k used, 2632k free, 53616k buffers  
Swap: 401584k total, 988k used, 400596k free, 20368k cached  
  
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  
6394 mgauglitz 16 0 920 920 744 R 0.7 0.7 0:00.07 top  
6405 mgauglitz 18 0 660 660 576 S 0.7 0.5 0:00.02 sleep  
6406 mgauglitz 18 0 660 660 576 S 0.7 0.5 0:00.02 sleep  
11698 root 15 0 59844 10m 1996 S 0.3 8.3 2:36.68 X  
1 root 15 0 100 88 64 S 0.0 0.1 0:05.82 init  
2 root 15 0 0 0 0 S 0.0 0.0 0:00.87 keventd  
3 root 15 0 0 0 0 S 0.0 0.0 0:00.03 kapmd  
4 root 34 19 0 0 0 S 0.0 0.0 0:00.05 ksoftirqd_CPU0  
5 root 16 0 0 0 0 S 0.0 0.0 0:02.92 kswapd  
6 root 25 0 0 0 0 S 0.0 0.0 0:00.00 bdflush  
7 root 15 0 0 0 0 S 0.0 0.0 0:25.13 kupdated  
8 root 16 0 0 0 0 S 0.0 0.0 0:00.10 kinoded  
9 root 25 0 0 0 0 S 0.0 0.0 0:00.00 mdrecoveryd  
12 root 15 0 0 0 0 S 0.0 0.0 0:05.20 kreiserfsd  
853 root 15 0 572 572 480 S 0.0 0.5 0:00.27 syslogd  
856 root 15 0 1316 1316 376 S 0.0 1.0 0:00.22 klogd  
915 root 19 0 0 0 0 S 0.0 0.0 0:00.00 khubd
```

# Zusammenfassung Kapitel 4

---

- In der Praxis schaltet die CPU zwischen den einzelnen Prozessen hin und her und erzeugt so die Illusion einer Parallelität.
- Um dieses Hin- und Herschalten zu ermöglichen, benötigt man ein Prozessmodell. Im Prozessmodell besitzt jeder Prozess seine eigene virtuelle CPU.
- Ein Prozess ist ein Programm in Ausführung inklusive der aktuellen Werte seines Befehlszählers, der Registerinhalte, der Variablenbelegungen, des Stack, des Zustands der geöffneten Dateien und Netzwerkverbindungen, der Programmdaten und der Speicherinformationen.
- Ein Prozess besteht aus drei Speichersegmenten: Text, Data und Stack
- In einem System wird aus den Prozessen eine Hierarchie aufgebaut, d.h. jeder Prozess besitzt einen Parentprozess, von dem er erzeugt wurde.
- Man unterscheidet in der Theorie drei Prozesszustände Running, Ready und Blocking, in der Praxis können diese Zustände feiner unterteilt sein.
- Die Prozesstabelle ist die Implementierung des Prozessmodells.
- Threads sind "Ausführungsfäden", d.h. sie laufen im Adressraum eines Prozesses und benötigen weniger Verwaltungsinformationen als Prozesse.
- Multithreading ist das schnelle Hin- und Herschalten zwischen den Threads eines Prozesses zur Erzeugung der Illusion einer Parallelität. Dies ist vergleichbar dem Hin- und Herschalten der CPU zwischen einzelnen Prozessen.

# Zusammenfassung Kapitel 4

---

- Threads verbessern die Strukturierung eines Programmes, sind leichter (schneller) zu erstellen und zu zerstören als Prozesse, können auf Multiprozessorsystemen echt parallel verarbeitet werden und erhöhen die Performance durch effiziente Überlappung von I/O-lastigen und CPU-lastigen Aktivitäten.
- Der Dispatcher ist verantwortlich für die Überführung der Prozesse in die verschiedenen Zustände. Der Scheduler wählt aus den rechenbereiten Prozessen, den aus, der als nächstes die Rechenzeit der CPU bekommt.
- Ein Scheduling-Algorithmus verfolgt teilweise für alle Arten von Systemen die gleichen Ziele, teilweise je nach Art des Systems verschiedene Ziele.
- Man unterscheidet unterbrechende (präemptive) und nicht unterbrechende (nicht präemptive) Scheduler.
- Nicht präemptive Scheduling-Algorithmen für Batch-Systeme sind First Come First Served (FCFS) und Shortest Job First (SJF). SJF ist beweisbar optimal, aber nicht realisierbar.

# Zusammenfassung Kapitel 4

---

- Präemptive Scheduling-Algorithmen für interaktive Systeme sind Round-Robin- Scheduling und Priority Based Scheduling.
- Das Quantum ist die Zeitspanne, für die ein Prozess die CPU zum Rechnen bekommt. Die Wahl des Quantums muss ein Kompromiss aus CPU-Effizienz und Antwortzeit eines Systems sein.
- Prozesse werden unter UNIX über eine residente Prozesstabellen und eine Benutzerstruktur realisiert.
- Prozesse unter UNIX entstehen immer aus anderen Prozessen, so dass eine Prozesshierarchie entsteht.
- Jeder Prozess hat eine UID, EUID, eine Standardeingabe, Standardausgabe und Standardfehlerausgabe.
- Prozesse unter UNIX können als Hintergrundprozesse laufen.
- Unter UNIX wird Priority Based Scheduling als CPU-Scheduler realisiert.
- Die Kommandos ps und top dienen dem Prozessmonitoring.

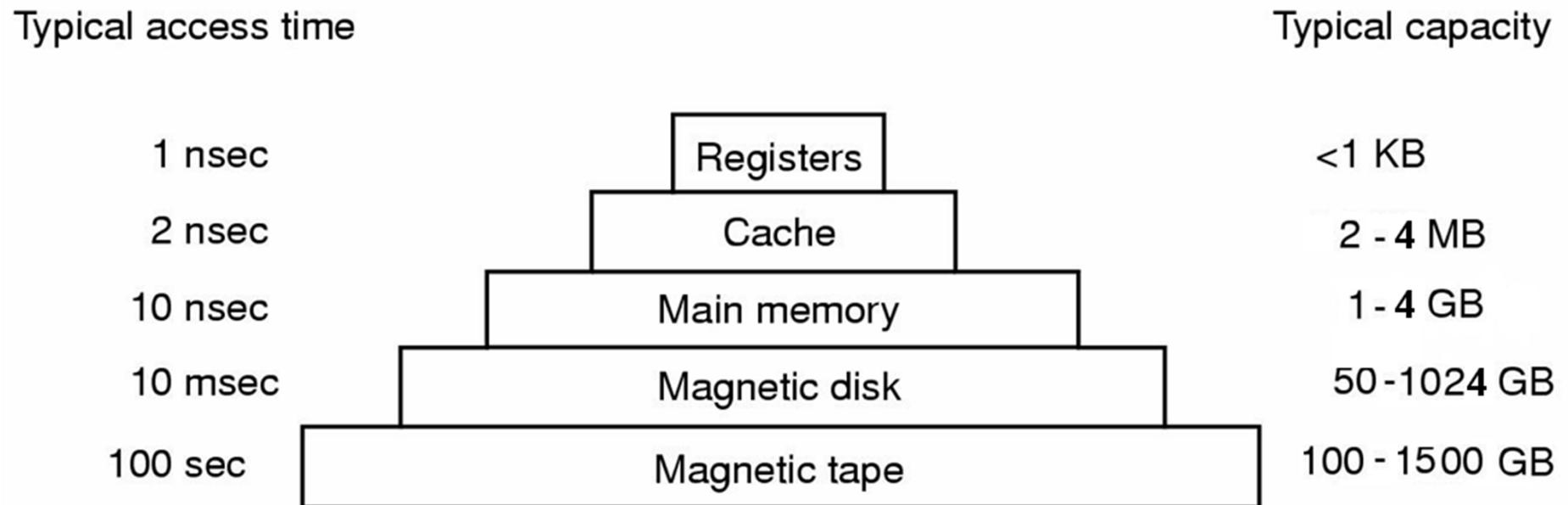
# Kapitel 5 – BS-Konzepte: Speichermanagement

---

- 5.1 Grundlagen des Speichermanagement**
- 5.2 Virtueller Speicher**
- 5.3 Seitenersetzungsalgorithmen**
- 5.4 Speichermanagement unter UNIX/Linux**

# 5.1 Grundlagen des Speichermanagement

- ideal: sehr schnell, sehr groß, sehr billig (nicht realisierbar !!!)
- deswegen: Speicherhierarchie (Kosten / Bit sinken mit der Kapazität)  
=> Verwaltung der Speicherhierarchie durch Speichermanagement



# 5.2 Virtueller Speicher

## Virtueller Speicher

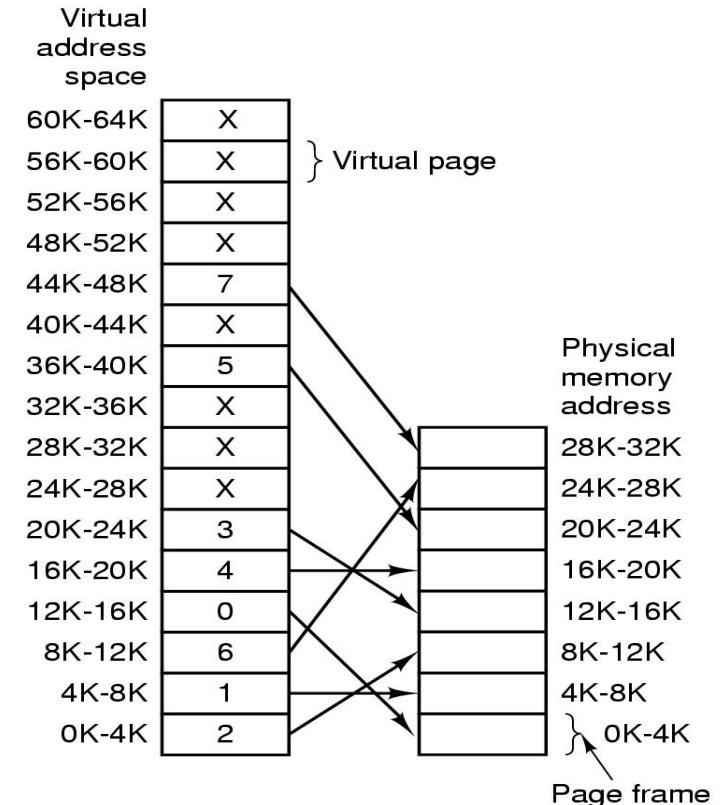
Programme zu groß für verfügbaren Speicher

-> Virtual Memory

- nur die aktuell benötigten Programmteile sind im Speicher
- der Rest ist auf Festplatte ausgelagert

### Technik Paging

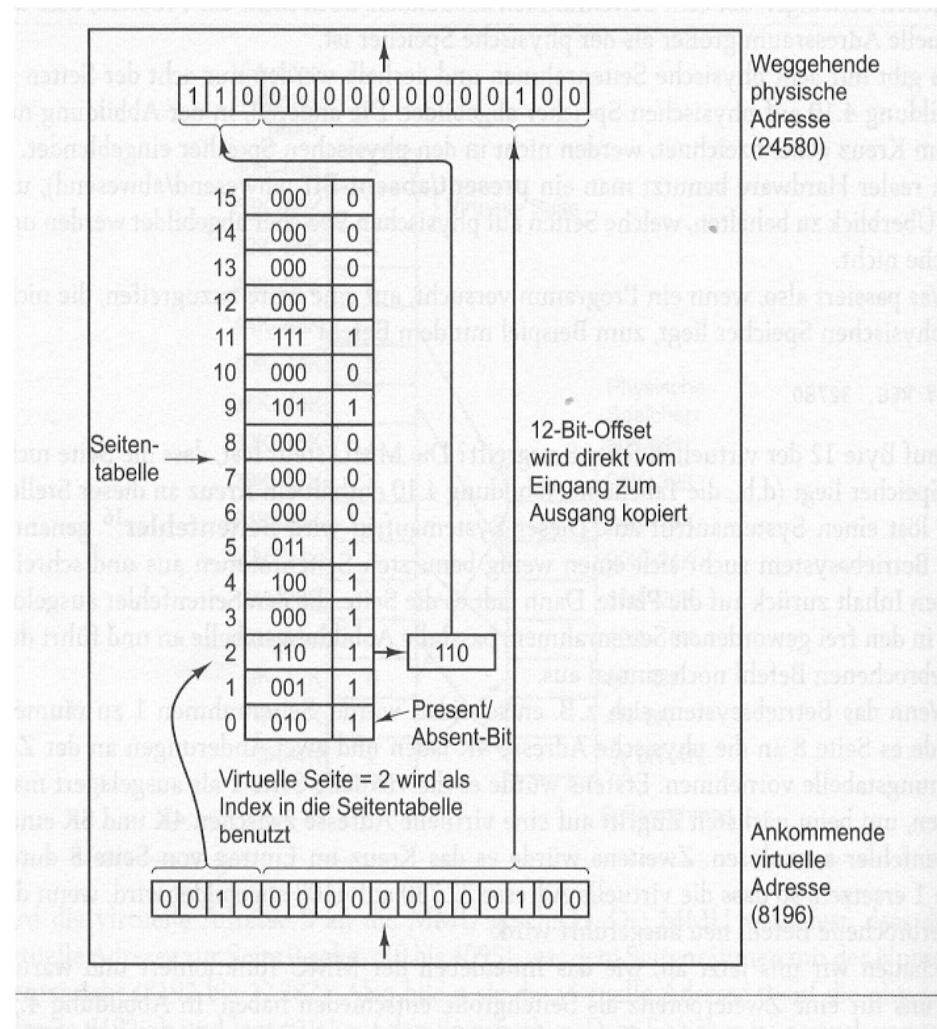
- Verwendung virtueller Adressen -> virtueller Adressraum
- MMU (Memory Management Unit) zur Abbildung virtueller Adressen auf physische Adressen
- Virtueller Adressraum in Seiten (pages) unterteilt
- Physischer Speicher: Seitenkacheln (page frames)
- Seiten und Seitenkacheln sind immer gleich groß
- zwischen Speicher und Platte werden immer ganze Seiten übertragen
- in realen Systemen zwischen 512 Byte und 64KB
- hier 4KB, 64KB virtueller und 32 KB physischer Adressraum
- hier 16 Seiten und 8 Seitenkacheln
- Seitentabelle enthält Zuordnung virtuelle Seite zu Seitenkachel
- Zugriff auf nicht zugeordnete Seite:
  - MMU: Systemaufruf Seitenfehler (page fault)
- wenig benutzte Kachel ausgelagert
- Seite in frei gewordene Kachel einlagern
- Abbildung anpassen
- nochmalige Befehlsausführung



# 5.2 Virtueller Speicher

## Funktionsweise der MMU:

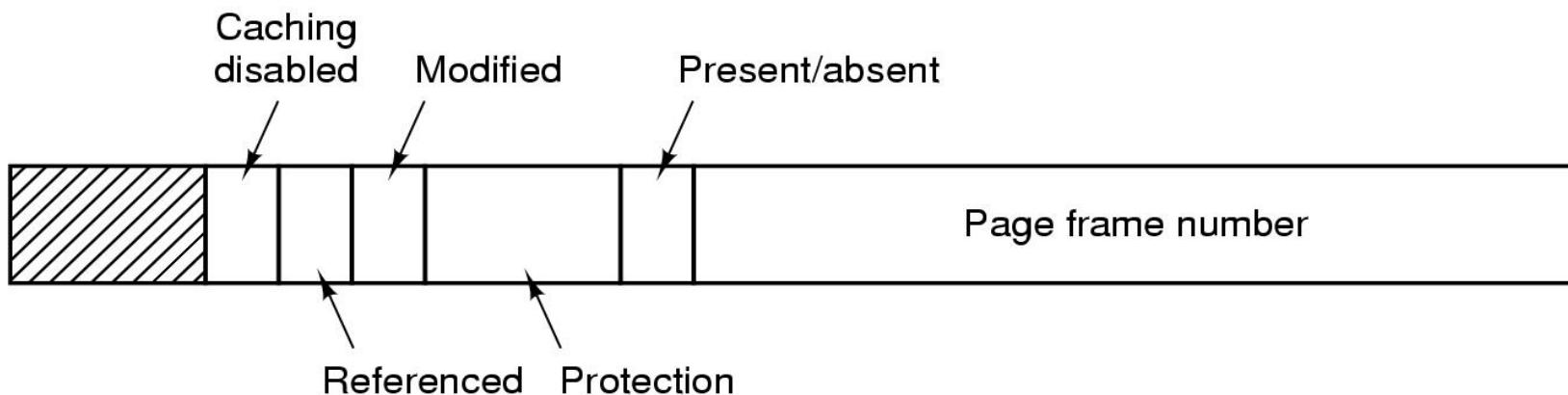
- virtuelle 16 Bit-Adresse wird zerlegt in
  - 4 Bit für Seitennummer (16 Seiten)**
  - 12 Bit Offset innerhalb der Seite (4096 Byte einer Seite)**
- Seitennummer wird als Index für Seitentabelle (page table) verwendet
- Seitentabelle enthält die Nummer der Seitenkachel, die der virtuellen Seite entspricht
- Wenn present/absent-Bit 0 → Page fault
- Sonst: Nummer der Seitenkachel vor den Offset  
-> 15 Bit große physische Adresse wird auf Speicherbus gelegt



# 5.2 Virtueller Speicher

## Seitentabellen (Page tables)

- Abbildung der virtuellen Seiten auf physische Seitenkacheln
- Einfachster Fall auf vorheriger Folie beschrieben
- mathematische Funktion.: IN: virtuelle Seitennummer, OUT: Seitenkachelnummer
- Probleme: Seitentabelle kann extrem groß werden und Umrechnung muss schnell sein
- Beispiel: virtuelle Adressen moderner Computer 32 Bit, Seitengröße 4KB
  - > 20 Bit für Seitentabelle, mehr als eine Million Seiten
- pro Befehl mehrere Zugriffe auf Seitentabelle, Befehl 4ns -> Tabellenzugriff max. 1ns



- Seitenkachelnummer : Nummer der Seitenkachel im Memory
- Present/Absent: Seite gerade im Memory oder nicht
- Protection: Lese-, Schreib-, Ausführungsrecht (1-3 Bit)
- Modified: ist die Seite inzwischen im Speicher beschrieben worden (M-Bit)
- Referenced: auch Lesezugriffe auf Seite werden protokolliert (R-Bit)
- Caching: bei Zugriff auf Geräteregister wichtig; bestimmt, ob auf Cache oder Register zugegriffen wird

# 5.3 Seitenersetzungsalgorithmen

---

## Seitenersetzungsalgorithmen

- Algorithmus nötig, der bei einem Page fault entscheidet, welche Seite aus dem Speicher entfernt wird, um Platz für die neue Seite zu machen
- Wenn auszulagernde Seite modifiziert wurde, muss sie auf Festplatte zurückgeschrieben werden
- Erster Ansatz: Beliebige Seite auslagern; besser: selten benutzte Seiten auslagern

Optimaler Algorithmus:

- Jede Seite wird mit der Anzahl Befehle markiert, die bis zum nächsten Zugriff auf diese Seite ausgeführt werden.
  - Entfernung der Seite mit der höchsten Zahl von Befehlen bis zum nächsten Zugriff
- => theoretisch sehr einfach, praktisch unmöglich  
=> Betriebssystem kann nicht wissen, wann auf welche Seite das nächste Mal zugegriffen wird

# 5.3 Seitenersetzungsalgorithmen

---

## First In First Out (FIFO)

- Betriebssystem verwaltet Liste aller Seiten im Speicher
- Am Kopf der Liste steht die älteste Seite, am Ende die zuletzt eingelagerte
- Bei Page Fault wird Seite am Kopf der Liste entfernt und die neue Seite ans Ende angehängt

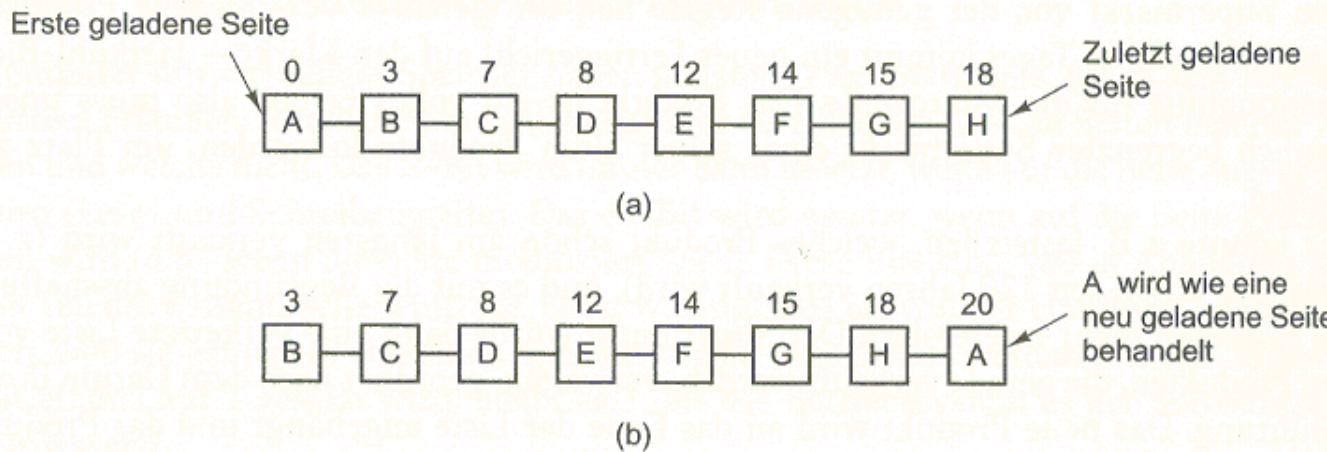
=> Nachteil: wichtige Seiten könnten entfernt werden

# 5.3 Seitenersetzungsalgorithmen

## Second Chance

- Einfache Variante von FIFO
  - R-Bit der ältesten Seite wird überprüft
  - Nicht gesetzt: alt und unbenutzt => sofortige Ersetzung
  - Sonst: R-Bit wird gelöscht und Seite ans Ende der Liste verschoben
- => enorme Verbesserung von FIFO, aber ähnlich einfach zu implementieren
- => Nachteil: unnötig ineffizient, durch ständiges Verschieben von Seiten in der Liste

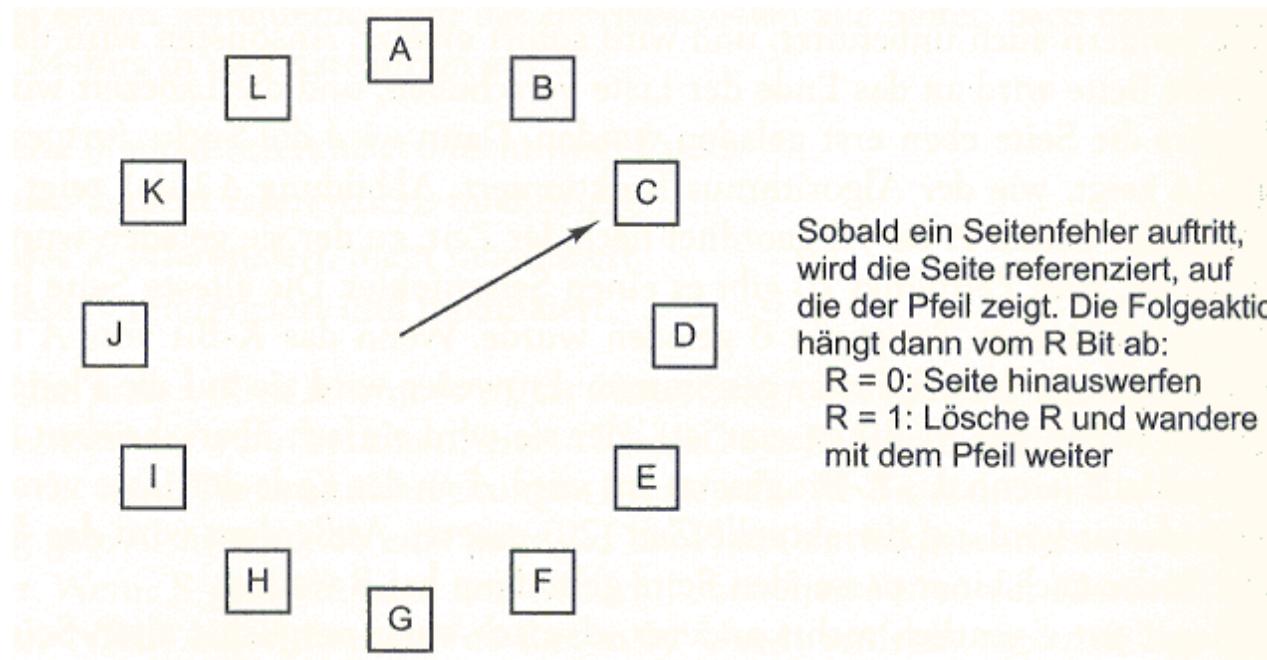
**Abbildung 4.16** Arbeitsweise von Second Chance. (a) Seiten in FIFO-Ordnung sortiert. (b) Die Seitenliste nach einem Seitenfehler zum Zeitpunkt 20, falls bei A das R-Bit gesetzt war. Die Zahlen sind Ladezeiten.



# 5.3 Seitenersetzungsalgorithmen

## Clock

- Vermeidet unnötige Verschiebungen von Seiten in der Liste durch ringförmige Anordnung und einen Uhrzeiger, der auf die älteste Seite zeigt



# 5.3 Seitenersetzungsalgorithmen

---

## Not Recently Used (NRU)

- R- und M-Bit werden verwendet
  1. Bei Prozessstart alle R- und M-Bits auf 0 setzen
  2. in bestimmten Zeitabständen alle R-Bits löschen (nur kürzlich benutzte Seiten haben R-Bit gesetzt)
  3. Bei Page fault Aufteilung der Seiten nach Zustand der R- und M-Bits
    - Klasse 0: R=0, M=0
    - Klasse 1: R=0, M=1
    - Klasse 2: R=1, M=0
    - Klasse 3: R=1, M=1
  4. Entfernung einer Seite aus der niedrigsten nicht leeren Klasse

=> leicht verständlich

=> effizient zu implementieren

=> keine optimale Leistung, in vielen Fällen ausreichend

# 5.3 Seitenersetzungsalgorithmen

## Least Recently Used (LRU):

- Entferne bei einem Page Fault die Seite, die am längsten unbenutzt ist
  - bei jeder Benutzung einer Seite wird die Seite an den Anfang der Liste aller Seiten verschoben, so dass die am längsten unbenutzte Seite am Ende der Liste steht
- => realisierbar, aber teuer (aufwendig)
- => verkettete Liste von allen Seiten im Speicher notwendig, Liste muss bei jedem Speicherzugriff aktualisiert werden, Verschiebung einer Seite an den Anfang der Liste aufwendig

## weitere Algorithmen:

- NFU (Not Frequently Used)
  - Aging
  - Working set
  - WSClock (am weitesten verbreitet)
- => nachlesbar im Tanenbaum

# 5.4 Speichermanagement unter UNIX/Linux

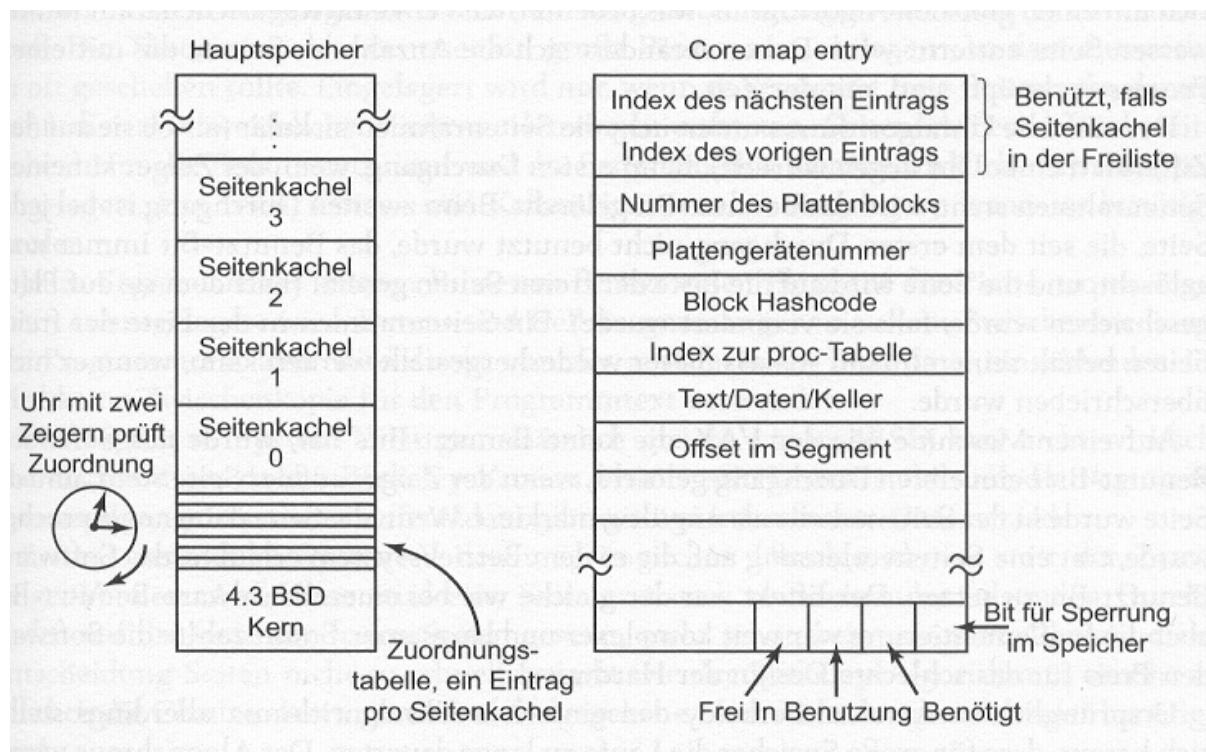
## Paging in UNIX

- grundlegende Idee:

Prozess muss nicht vollständig im Speicher sein, um ausgeführt zu werden

=> nur Benutzerstruktur und Seitentabellen benötigt, Seiten dynamisch einlagern

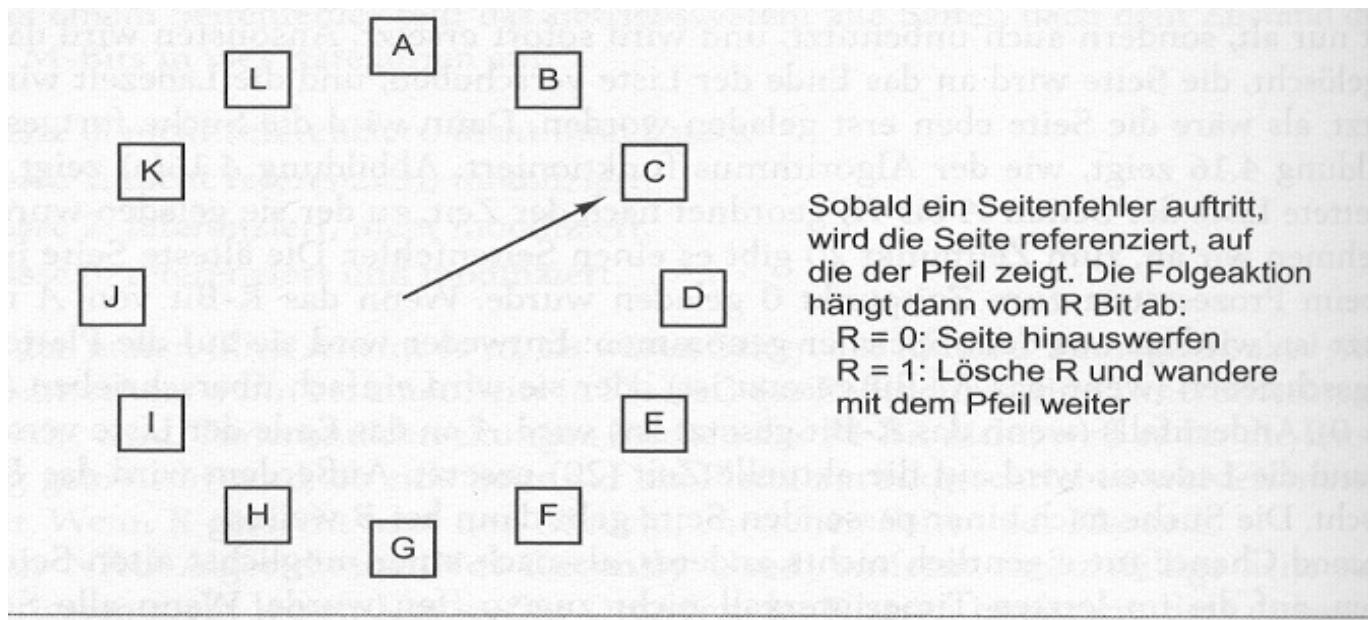
- Paging durch Kernel und einen Prozess namens Page Daemon realisiert



# 5.4 Speichermanagement unter UNIX/Linux

## Seitenersetzungsalgorithmus

- wird vom Page Daemon ausgeführt
- alle 250ms Überprüfung, ob Anzahl freie Seiten  $\geq \text{lotsfree}$  (meist 25% Hauptspeicher)
- Verschiebung Seitenkacheln auf Platte, bis  $\text{lotsfree}$  frei
- ursprünglich Clock-Algorithmus verwendet (siehe Kap. 5.3),
- ähnlich Second Chance (siehe Kap. 5.3), keine lineare Liste, sondern ringförmige Liste  
Zeiger zeigt auf älteste Seite => vermeidet Verschieben ans Ende der Liste



=> für groÙe Speicher, Zeigeraute zu lange

# 5.4 Speichermanagement unter UNIX/Linux

---

## Zwei-Zeiger-Clock-Algorithmus

- Löschen des R-Bit am vorderen Zeiger
- Prüfen R-Bit des hinteren Zeigers (wie im Clock-Algorithmus)
- Weitersetzen der Zeiger

wenn Zeiger nahe zusammen: nur häufig benutzte Seiten bleiben im Speicher

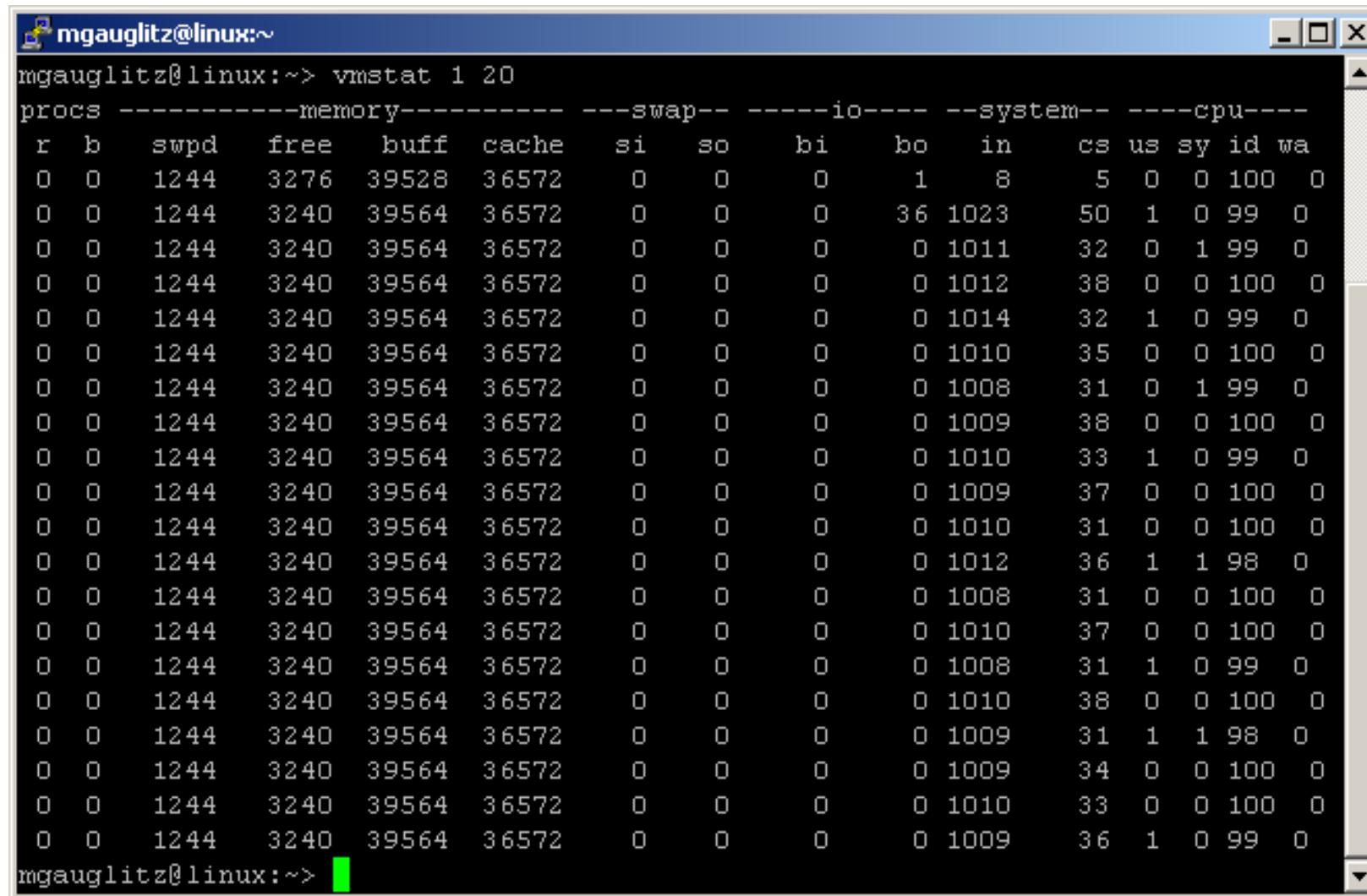
wenn Zeiger 359 Grad auseinander: Clock-Algorithmus

# 5.4 Speichermanagement unter UNIX/Linux

## Speichermonitoring

vmstat liefert Infos über den Virtual Memory

Bsp. Vmstat 1 20 (jede Sekunde eine Ausgabe, 20mal)



The screenshot shows a terminal window titled "mgauglitz@linux:~". The command "vmstat 1 20" is run, displaying a continuous loop of system monitoring data. The output is a table with the following columns:

procs		memory					swap		io			system				cpu			
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa				
0	0	1244	3276	39528	36572	0	0	0	1	8	5	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	36	1023	50	1	0	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1011	32	0	1	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1012	38	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1014	32	1	0	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	35	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1008	31	0	1	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1009	38	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	33	1	0	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1009	37	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	31	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1012	36	1	1	98	0				
0	0	1244	3240	39564	36572	0	0	0	0	1008	31	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	37	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1008	31	1	0	99	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	38	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1009	31	1	1	98	0				
0	0	1244	3240	39564	36572	0	0	0	0	1009	34	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1010	33	0	0	100	0				
0	0	1244	3240	39564	36572	0	0	0	0	1009	36	1	0	99	0				

## 5.4 Speichermanagement unter UNIX/Linux

---

Weitere Kommandos zum Speichermonitoring:

- free
- top

# Zusammenfassung Kapitel 5

---

- Der Hauptspeicher ist nicht groß genug, um alle gerade benötigten Programme aufnehmen zu können. Dieses Problem kann mit virtuellem Speicher gelöst werden.
- Beim virtuellen Speicher werden nur die gerade benötigten Teile eines Prozesses in den Hauptspeicher eingelagert und der Rest des Prozesses ist auf die Festplatte ausgelagert.
- Der virtuelle Speicher wird in Seiten eingeteilt, der physische Speicher in Seitenkacheln. Die Seitentabelle verwaltet die Zuordnung der Seiten zu den Seitenkacheln. Wird auf eine virtuelle Seite, die keine Zuordnung zu einer Seitenkachel besitzt, referenziert, kommt es zu einem Page Fault.
- Die Zuordnung von virtueller Adresse zu physischer Adresse ist durch eine mathematische Funktion beschreibbar. Die Memory Management Unit (MMU) übernimmt die Berechnung dieser Funktion, d.h. die Umrechnung von virtueller in physische Adresse.
- Bei einem Page Fault entscheidet ein Seitenersetzungsalgorismus, welche Seitenkachel ausgelagert wird (am besten eine wenig benutzte), um Platz für die einzulagernde Seitenkachel zu schaffen.

# Zusammenfassung Kapitel 5

---

- Beispiele für Seitenersetzungsalgorithmen sind NRU, Second Chance und Clock.
- Paging wird unter UNIX durch den Kernel und einen Prozess namens Page Daemon implementiert. Die eingesetzten Seitenersetzungsalgorithmen sind Varianten des Clock-Algorithmus.
- Die Kommandos vmstat, top und free können zum Speichermonitoring verwendet werden.

# Kapitel 6 – BS-Konzepte: Input/Output-Management

---

- 6.1 Grundlagen der Ein-/Ausgabegeräte**
- 6.2 Grundlagen der Software für Ein-/Ausgabe**
- 6.3 Ein- und Ausgabegeräte**
- 6.4 Input/Output-Management unter UNIX/Linux**

# 6.1 Grundlagen der Ein-/Ausgabegeräte

---

## Ein-/Ausgabe (I/O):

- Eine der Hauptaufgaben des OS: Überwachung und Steuerung der I/O-Geräte
- Bereitstellung einer Schnittstelle zwischen Geräten und Rest des Systems:  
einfach benutzbar und geräteunabhängig
- Verschiedene Blickwinkel:
  - E-Techniker: Chips, Drähte, Motoren etc.
  - Programmierer: Schnittstelle zur Software (Kommandos, Funktionen, Fehler)
  - Benutzer: Bedienung und Verwendung der Geräte (Funktionalität)

# 6.1 Grundlagen der Ein-/Ausgabegeräte

## Grundsätzliches über Ein-/Ausgabegeräte

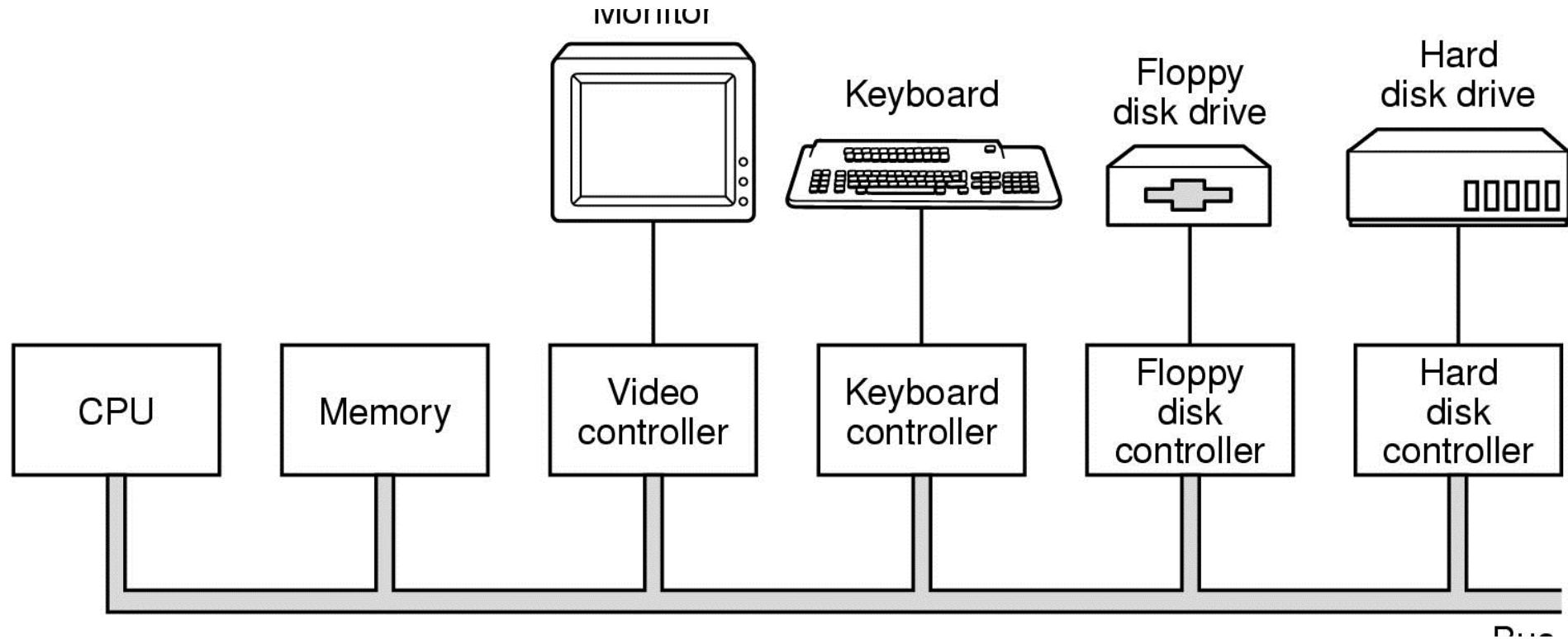
- grobe Einteilung:
  - blockorientierte Geräte (block devices), z.B. Festplatten (Blöcke fester Größe mit eigener Adresse, 512 bis 32768 Byte, jeder Block kann unabhängig von jedem anderen gelesen/geschrieben werden)
  - zeichenorientierte Geräte (character devices), z.B. Drucker (erzeugt/akzeptiert Zeichenströme, nicht adressierbar, keine Suche)

- typische Geräte und Transferraten:

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

# 6.1 Grundlagen der Ein-/Ausgabegeräte

- Teilung eines Gerätes in mechanischen und elektronischen Teil (Adapter oder Controller)



- Controller können i.a. mehrere Geräte verwalten
- Aufgabe des Controllers: Datenkonvertierung (serieller Bitstrom-> Blöcke) in Puffer, Fehlerkorrektur (ECC), Kopieren in Speicher

# 6.1 Grundlagen der Ein-/Ausgabegeräte

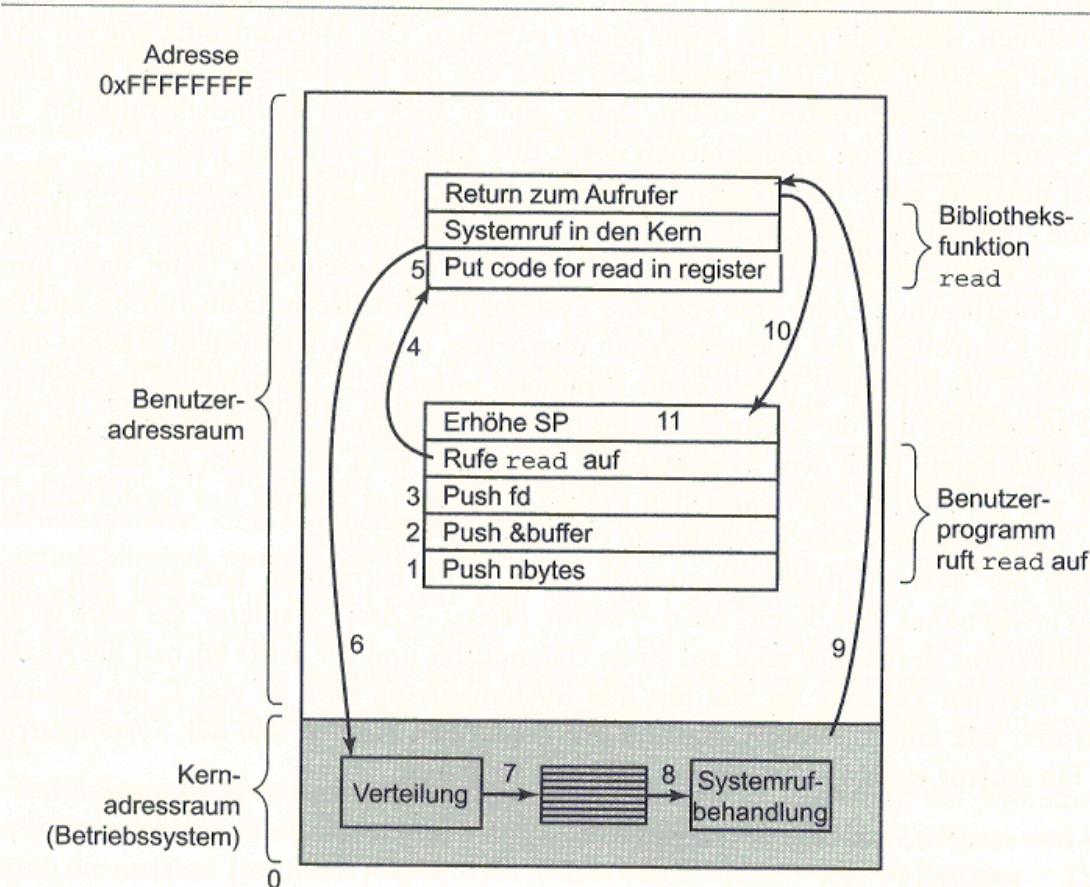
---

## Drei Arten zur E/A Behandlung

- Systemaufruf (CPU unnötig lange beschäftigt)
- Interrupt: Gerät wird beauftragt, OS macht inzwischen weiter
- Benutzung eines DMA Chips (Direct Memory Access): regelt Datenfluss zwischen Speicher und Controller (CPU wird nicht belastet)

# 6.1 Grundlagen der Ein-/Ausgabegeräte

Abbildung 1.17 11 Schritte für den Systemaufruf `read(fd, buffer, nbytes);`.



⇒ CPU während Systemaufruf ständig beschäftigt  
(busy waiting)

**Beispiel: Systemaufruf read**

**1 – 3 übergebene Parameter auf Stack**

**4 Sprung in Bibliotheksfunktion read**

**5 Bibliotheksfunktion speichert die Nummer des Systemaufrufes read in Register**

**6 TRAP (Assemblerbefehl) für Sprung in den Kernelmode**

**7 Systemaufrufnummer als Index für Tabelle, in der alle Systemaufrufe stehen**

**8 Systemaufruf read**

**9 nach Ende des Systemaufrufes read Sprung zurück in Bibliotheksfunktion read**

**10 nach Ende der Bibliotheksfunktion Rücksprung zum aufrufenden Programm**

**11 Stack aufräumen durch Erhöhung (Stack wächst nach unten) oder Erniedrigung (Stack wächst nach oben) des Stackpointers**

# 6.1 Grundlagen der Ein-/Ausgabegeräte

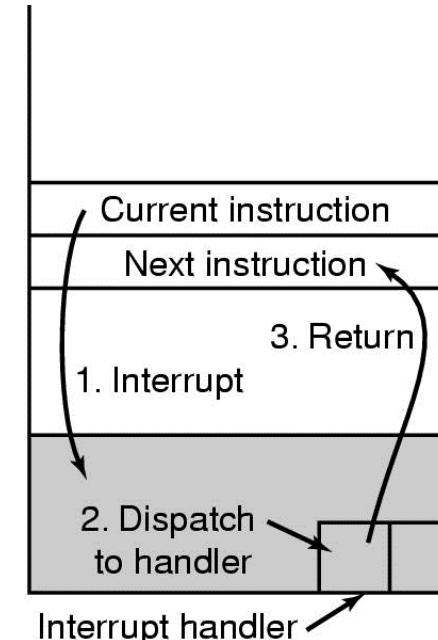
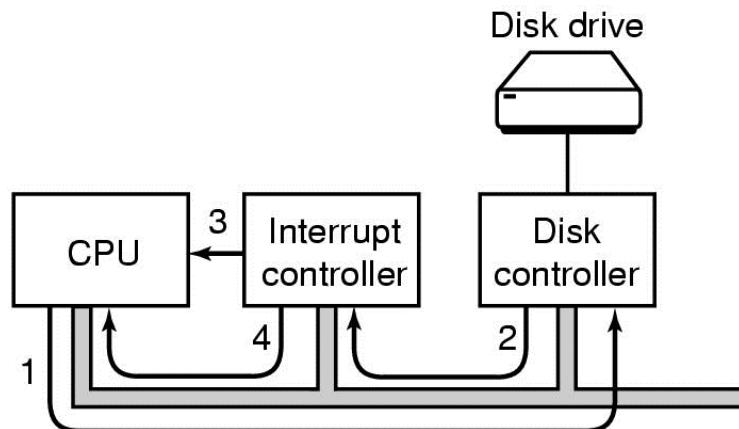
Interruptbehandlung:

linkes Bild:

1. Treiber an Controller (was soll Gerät tun)
2. Gerät fertig, Controller an Interruptcontroller (Signal)
3. Falls Interruptcontroller bereit: Nachricht an CPU
4. Gerätenummer auf Bus zur Identifiz. durch CPU, CPU behandelt Interrupt (rechts)

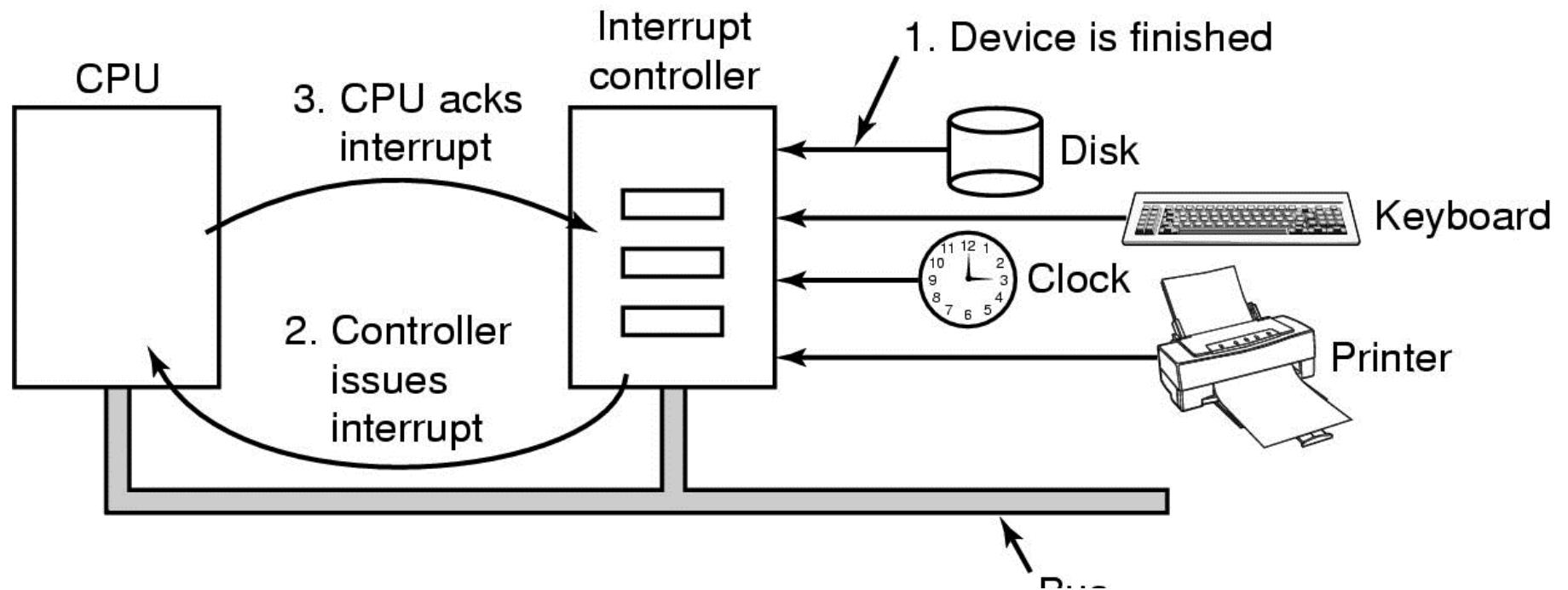
rechtes Bild:

1. Interruptanforderung, Programmstatus merken (Stack)
2. Sprung in Kernelmode, Interruptroutine starten (Daten an Programm)
3. Rücksprung in Usermode (Programm)



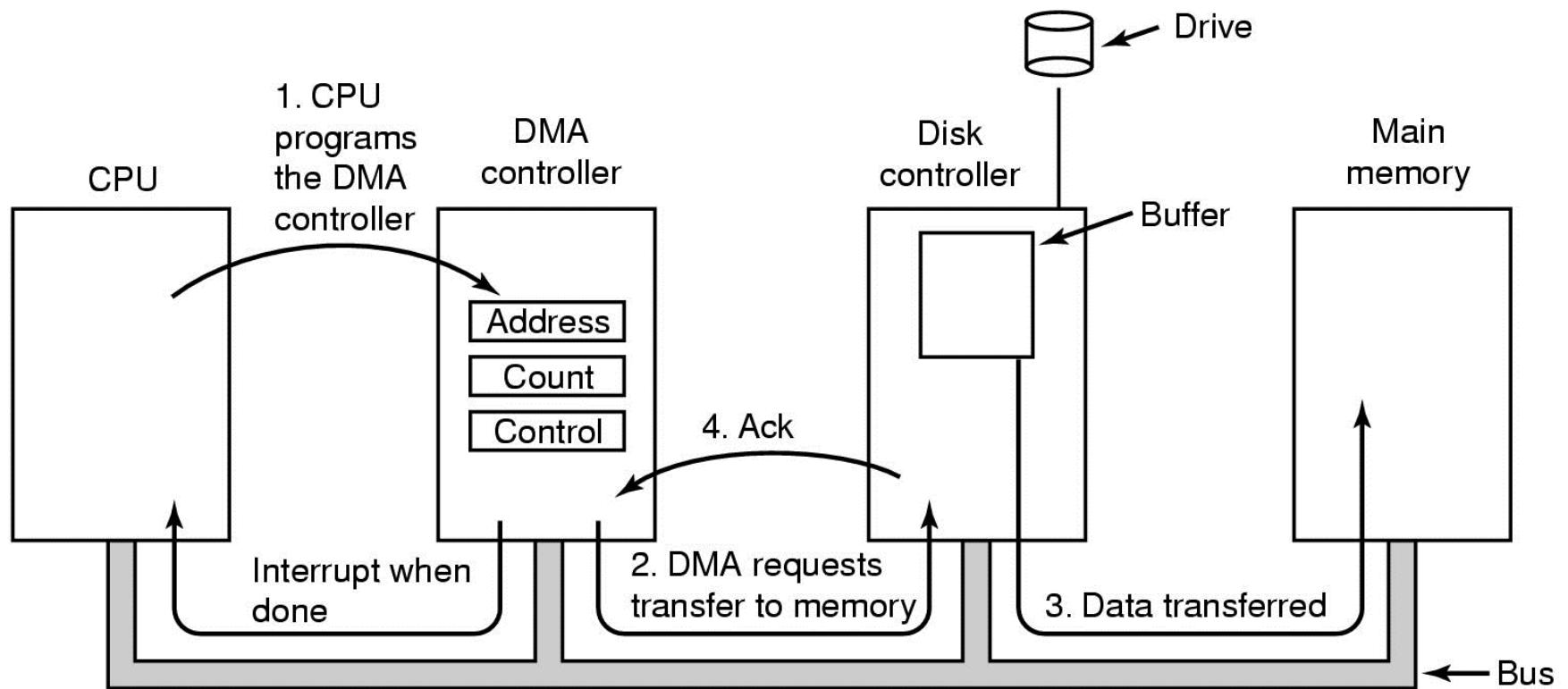
# 6.1 Grundlagen der Ein-/Ausgabegeräte

- Interruptbehandlung: hauptsächlich langsame Geräte arbeiten mit Interrupts



# 6.1 Grundlagen der Ein-/Ausgabegeräte

- Adressierung der Gerätecontroller mittels Direct Memory Access (DMA)
  - direkt im Gerätecontroller integriert -> jedes Gerät erfordert eigenen DMA-Controller
  - auf Hauptplatine



# 6.1 Grundlagen der Ein-/Ausgabegeräte

---

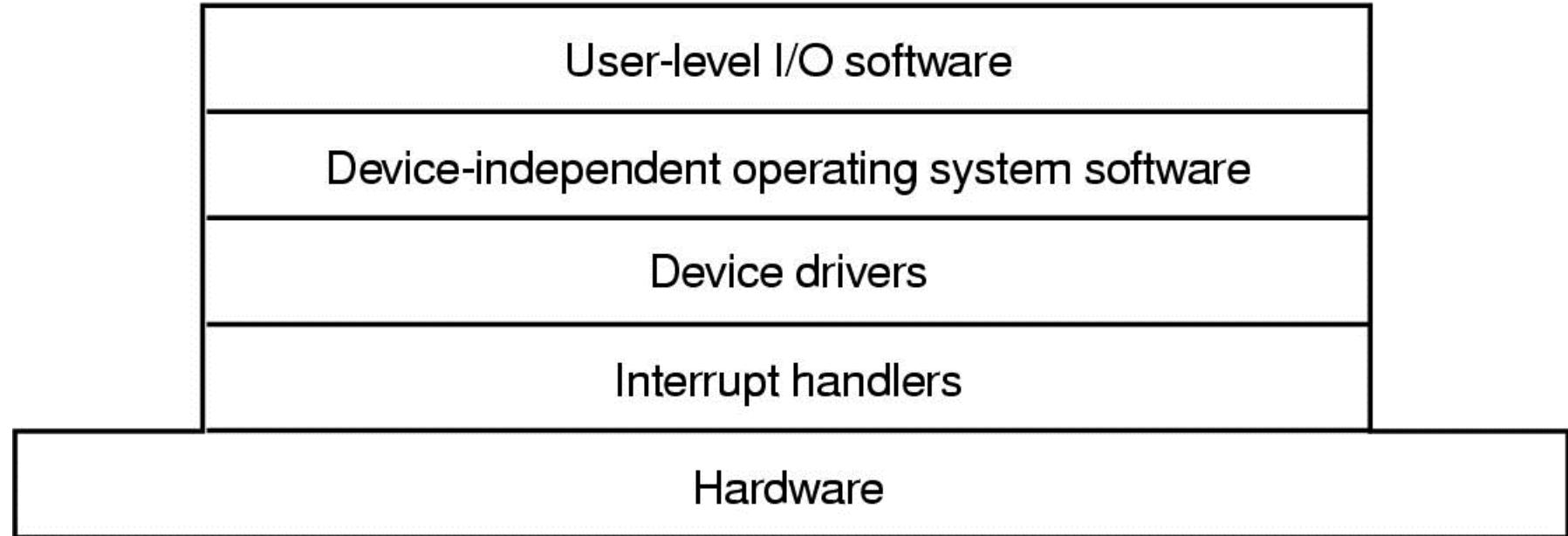
## Vier Schritte zum Ablauf einer Leseoperation:

1. - CPU setzt Register des DMA-Controllers (was soll wohin, d.h. u.a. Zielspeicheradresse und Anzahl zu übertragender Bytes)  
- CPU schickt Kommando zum Lesen an Plattencontroller  
- Plattencontroller führt das Kommando aus und prüft die Daten  
- bei gültigen Daten weiter mit 2.
2. - DMA-Controller sendet Lesebefehl für den Puffer des Plattencontrollers über den Bus -  
- legt Zielspeicheradresse des Hauptspeichers auf die Adressleitung des Busses
3. - Schreiben eines Wortes vom Puffer des Plattencontrollers in den Hauptspeicher in einem Standardbuszyklus
4. - Plattencontroller meldet ACKnowledge an DMA-Controller, wenn Wort übertragen  
- DMA-Controller erhöht Zielspeicheradresse, verringert Anzahl zu übertragender Bytes  
- Schritte 2.-4. werden wiederholt, solange Anzahl der zu übertragenden Bytes  $> 0$  ist  
- wenn  $=0$  führt DMA-Controller einen Interrupt aus, um der CPU die Beendigung der Datenübertragung mitzuteilen

=> Betriebssystem muss nun nichts mehr tun, weil Daten schon im Hauptspeicher vorhanden sind

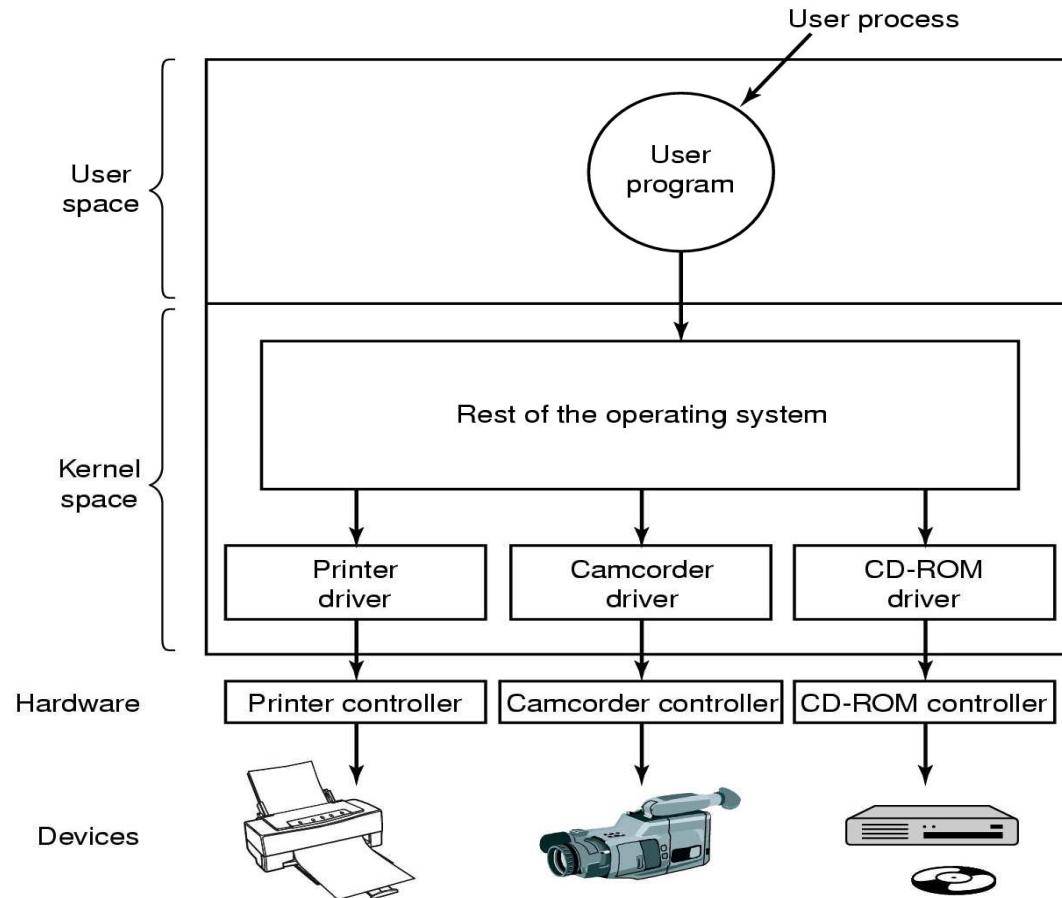
## 6.2 Grundlagen der Software für Ein-/Ausgabe

### Schichten der Ein-/Ausgabe



# 6.2 Grundlagen der Software für Ein-/Ausgabe

Positionierung der Gerätetreiber:



## 6.2 Grundlagen der Software für Ein-/Ausgabe

---

**Drei Möglichkeiten zur Integration von Gerätetreibern in Kernel:**

- statisch einbinden (linken): z.B. Unix
- Treibername in Datei, OS liest Datei beim Booten und lädt Treiber (Windows)
- Kernelmodule (dynamisch nachladbar), z.B. Linux, einige Unix-Derivate

## 6.2 Grundlagen der Software für Ein-/Ausgabe

---

Standardschnittstellen zur Benutzung von Treibern:

- für blockorientierte Geräte (z.B. Lesen eines Blocks)
- für zeichenorientierte Geräte (z.B. Schreiben eines Zeichenstroms)

Funktionen eines Gerätetreibers:

- Entgegennehmen von Lese-/Schreibbefehlen aus höher liegenden Schichten
- Überwachung der Ausführung
- Gerät initialisieren
- Energieverwaltung
- Ereignisverwaltung

# 6.3 Ein- und Ausgabegeräte

## Plattenspeicher:

- Magnetplatten (Festplatten, Disketten)
- optische Medien: CD-ROM, DVD-ROM, BluRay, beschreibbare CD, DVD
- Solid State Disks (SSD): eigentlich keine Platten, sondern Flashspeicher

## Magnetplatten

- Material: Aluminium, Metalllegierung oder Glasplatten mit 5.25", 3.5", 2.5" oder 1,8" Durchmesser
- Oberfläche: magnetisierbare Metalloxidschicht

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 $\mu$ sec

# 6.3 Ein- und Ausgabegeräte

---

## Vier Schritte zur Benutzung einer Magnetplatte

1. Low Level Formatierung (Einteilung der Platte in Sektoren)
  2. Erzeugung des Master Boot Record (Bootcode und Partitionstabelle)
  3. Partitionierung der Magnetplatte (Einträge in Partitionstabelle)
  4. High Level Formatierung einer Partition mit einem Dateisystem
- => Anschließend kann auf dem Dateisystem ein Betriebssystem installiert werden.

# 6.3 Ein- und Ausgabegeräte

## Schritt 1: Low Level Formatierung

Einteilung der Platte in Sektoren

Sektor besteht aus:

- Präambel: Verwaltungsinformationen (Markierung Sektorbeginn, Zylindernummer, Sektornummer)
- Datenteil: Größe durch Formatierungsprogramm festgelegt (meist 512 Byte)
- ECC (Error Correcting Code): redundante Informationen zur Wiederherstellung der Daten bei Lesefehlern, Größe abhängig von gewünschter Zuverlässigkeit, bis 16 Byte

Preamble	Data	ECC
----------	------	-----

Low-Level-Formatierung reduziert Kapazität (ca. 20% weniger) wegen

- Präambel
- Lücke zwischen Sektoren
- Fehlerkorrekturcode

Low-Level-Formatierung beeinflusst Geschwindigkeit:

Beispiel: Festplatte 10000 U/min, 300 Sektoren pro Spur zu je 512 Byte  
=> 6ms, um 153600 Byte einer Spur zu lesen  
=> 24,4 MB/s Datenrate  
=> nicht übertreffbar, egal welche Schnittstelle benutzt wird  
selbst wenn SCSI mit 80MB/s oder 160 MB/s

# 6.3 Ein- und Ausgabegeräte

---

## Schritt 2: Erzeugung Master Boot Record

- Sektor 0
- Bootcode
- Partitionstabelle (Startsektor und Größe jeder Partition)

## Schritt 3: Partitionierung

- Einteilung der Platte in mehrere Partitionen unterschiedlicher oder gleicher Größe

## Schritt 4: High Level Formatierung

- jede einzelne Partition getrennt voneinander
- schreibt Bootblock an Anfang der Partition
- Liste der freien Bereiche zur Speicherverwaltung
- Wurzelverzeichnis
- leeres Dateisystem
- Code in Partitionstabelle für benutztes Dateisystem

Einschalten Rechner

- => BIOS startet und liest Master Boot Record
- => Bootprogramm prüft aktive Partition
- => Bootsektor wird gelesen (enthält Programm, das Betriebssystem oder Bootloader sucht)

# 6.4 I/O-Management unter UNIX/Linux

---

## Ein-/Ausgabegeräte in UNIX

- Geräte über Spezialdateien in Dateisystem ansprechbar (z.B. /dev/hd1, /dev/lp etc.)
- normale read/write-Aufrufe funktionieren auf diesen Dateien
- blockorientierte Spezialdateien: Blöcke adressierbar (z.B. Festplatten)
- zeichenorientierte Spezialdateien: serieller Zeichenstrom (z.B. Tastatur, Drucker, Netzwerk, Mäuse, Plotter)
- mit jeder Spezialdatei ist ein Gerätetreiber verbunden
- jeder Treiber hat eine Hauptgerätenummer (Major number) zur Identifikation
- Treiber benutzt mehrere Geräte (z.B. Festplatten) => jedes Gerät hat Nebengerätenummer (Minor number) zur Identifikation

# 6.4 I/O-Management unter UNIX/Linux

---

## Implementierung der Ein-/Ausgabe in UNIX

- Ein-/Ausgabe mit einem Gerätetreiber pro Gerätetyp implementiert
- Aufgabe des Treibers ist Isolation des Rest des Systems von Eigenheiten der Hardware
- durch Bereitstellung von Schnittstellen zwischen Treibern und Rest des Systems kann der größte Teil des E/A-Systems in maschinenunabhängigen Teil des Kernels gelegt werden

Ansprechen einer Spezialdatei:

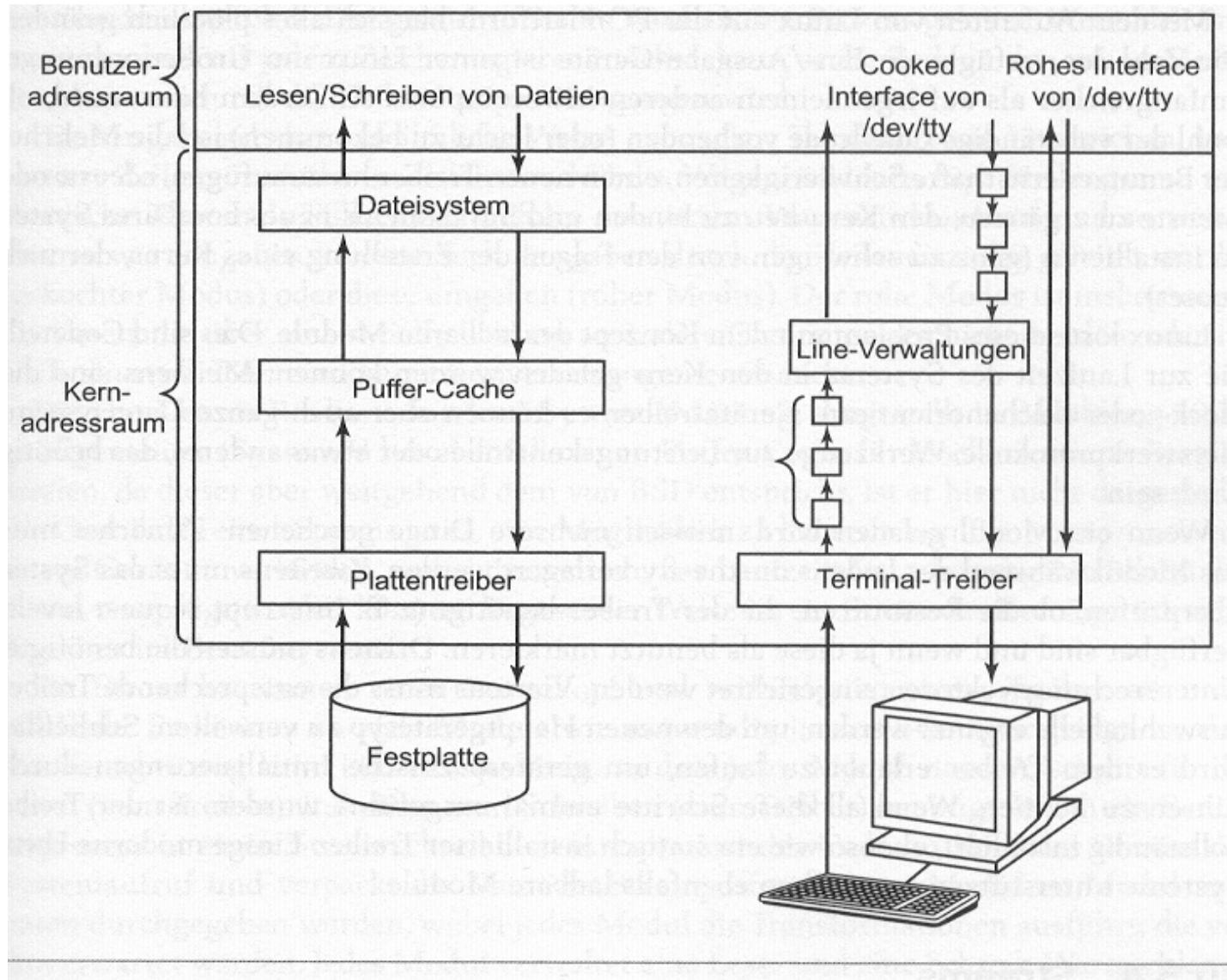
- System stellt Major und Minor Number fest und ob zeichen- oder blockorientiert
- Major number ist Index auf Struktur in Tabelle `bdevsw` oder `cdevsw`
- Struktur enthält Zeiger auf Prozeduren zum Öffnen/Lesen/Schreiben des Gerätes, Minor Number wird als Parameter übergeben
- Hinzufügen eines Gerätes => neuer Eintrag in Tabelle

Gerätetreiber:

- statisch in Kernel eingebunden (bei überschaubarer und selten veränderter Hardware)
- bei Linux und modernen UNIX-Systemen: ladbare Kernelmodule (Hardware variabel)

# 6.4 I/O-Management unter UNIX/Linux

## Das UNIX-Ein-/Ausgabe-System in BSD



# Zusammenfassung Kapitel 6

---

- Geräte können grob in zwei Gruppen eingeteilt werden: **blockorientierte und zeichenorientierte Geräte (block devices und character devices)**.
- Geräte bestehen aus einem mechanischen und einem elektronischen Teil. Den elektronischen Teil nennt man Adapter oder Controller.
- Es gibt drei Arten, wie Ein- und Ausgabe behandelt werden kann. Die einzelnen Arten unterscheiden sich durch den Grad der Belastung der CPU.
- DMA-Controller (DMA = Direct Memory Access) ermöglichen eine Datenübertragung vom Gerätetreiber zum Hauptspeicher und umgekehrt, die die CPU nicht belastet.
- Die Ein- und Ausgabe lässt sich in einem Schichtenmodell darstellen, in dem die benachbarten Schichten über definierte Schnittstellen miteinander kommunizieren.
- Gerätetreiber bieten einerseits **Standardschnittstellen für die geräteunabhängige Betriebssystem-Software**, andererseits steuern sie das Gerät über den Gerätetreiber an.

# Zusammenfassung Kapitel 6

---

- Da Gerätetreiber im Kerneladressraum laufen müssen (direkter Zugriff auf die Hardware), sind Gerätetreiber kritisch bezüglich der Stabilität des Systems. Da sie oftmals von Fremdherstellern geliefert werden, müssen die Interaktion mit dem Rest des Systems und die Architektur zur Installation des Codes im Kernel exakt spezifiziert sein.
- Es gibt drei Arten zur Integration von Gerätetreibern in den Betriebssystemkern. Die Arten unterscheiden sich durch die Flexibilität bezüglich Hardwareänderungen im System.
- Festplatten werden zunächst Low-Level-formatiert, anschließend wird der Master Boot Record erzeugt, dann werden eine oder mehrere Partitionen erstellt und zuletzt wird jede einzelne Partition der Festplatte mit einem Dateisystem High-Level-formatiert.
- Ein- und Ausgabe wird in UNIX durch Gerätespezialdateien implementiert. Man unterscheidet zeichenorientierte und blockorientierte Gerätespezialdateien. Jede Gerätespezialdatei besitzt eine Major und Minor Number.

# **Kapitel 7 – BS-Konzepte: Dateisysteme**

---

**7.1 Dateien**

**7.2 Verzeichnisse**

**7.3 Das UNIX-Dateisystem**

# 7.1 Dateien

---

## Anforderung an die langfristige Datenhaltung

- Speicherung großer Informationsmengen
  - Information überlebt Terminierung des erzeugenden Prozesses
  - mehrere Prozesse können gleichzeitig die Informationen nutzen
- 
- ⇒ Unabhängigkeit von Entstehung und Beendigung eines Prozesses
- ⇒ Speicherung der Daten in Dateien
- ⇒ Persistenz (dauerhafte Speicherung)

Dateisystem: Teil des Betriebssystems, der sich mit Dateien befasst

# 7.1 Dateien

---

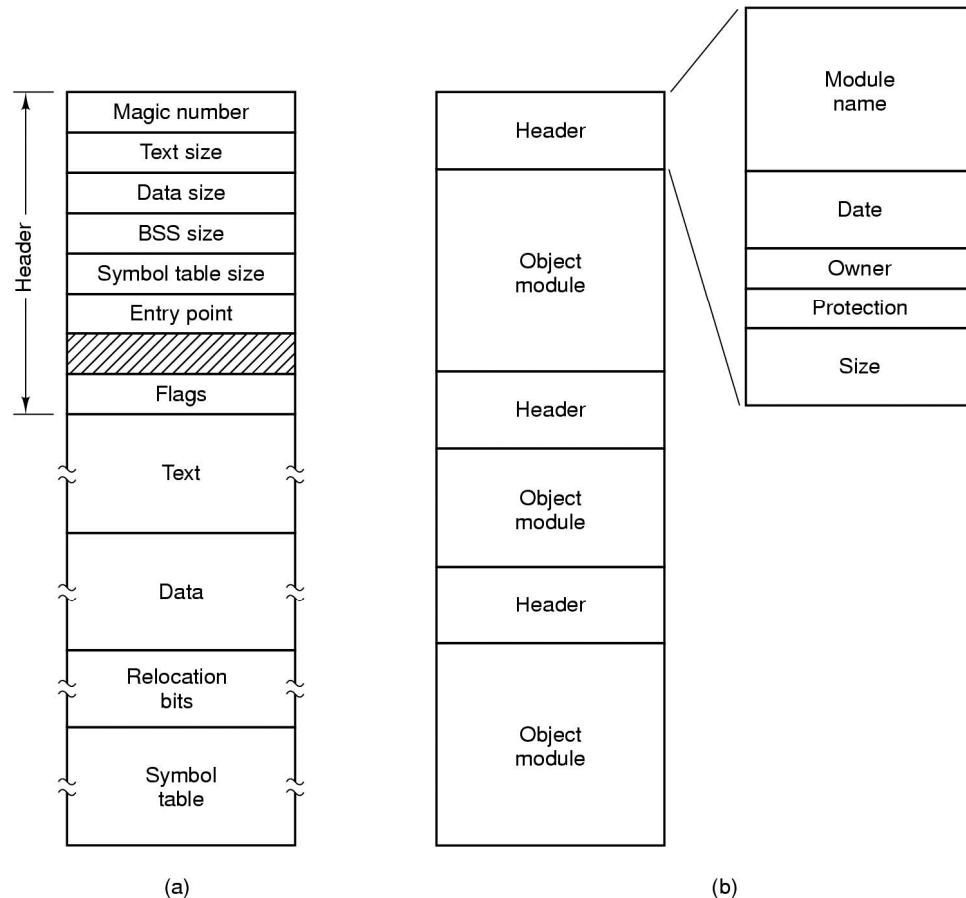
## Benennung von Dateien

- Dateiname zum Zugriff auf eine Datei
- Unterscheidung Groß- und Kleinschreibung (bei UNIX, bei MS-DOS nicht)
- Windows 95/98 benutzen MS-DOS-Dateisystem, NT und Win2k unterstützen MS-DOS, haben aber eigenes Dateisystem NTFS, Windows XP, Windows Vista und Windows 7 benutzen NTFS
- Oftmals Zweiteilung der Namen durch einen Punkt: Teil nach Punkt heißt Dateierweiterung (Extension)
  - => Dateierweiterung wird beim Betriebssystem registriert, Doppelklick bewirkt Start eines bestimmten Programms (Windows)
- MS-DOS: 8 + 3, Unix: Name bis 255 Zeichen, mehrere Extensions möglich

# 7.1 Dateien

## Dateitypen

- reguläre Dateien: ASCII- oder Binärdateien ((a) ausführbare Datei, (b) Archiv)
- Verzeichnisse: Systemdateien, um Dateisystemstruktur zu verwalten
- spezielle Zeichendateien: für serielle E/A Geräte, z.B. Drucker, Terminals, Netzwerke
- spezielle Blockdateien: für Massenspeicher



# 7.1 Dateien

---

## Dateizugriff

- **sequentieller Zugriff** (**Lesen aller Bytes einer Datei nacheinander, beginnend am Anfang**)
- **wahlfreier Zugriff** (**random access, in beliebiger Reihenfolge**)
  - 2 Methoden:** **read**, bei jedem Lesen wird Position in Datei festgelegt
  - seek, Position festlegen, dann sequentiell lesen**

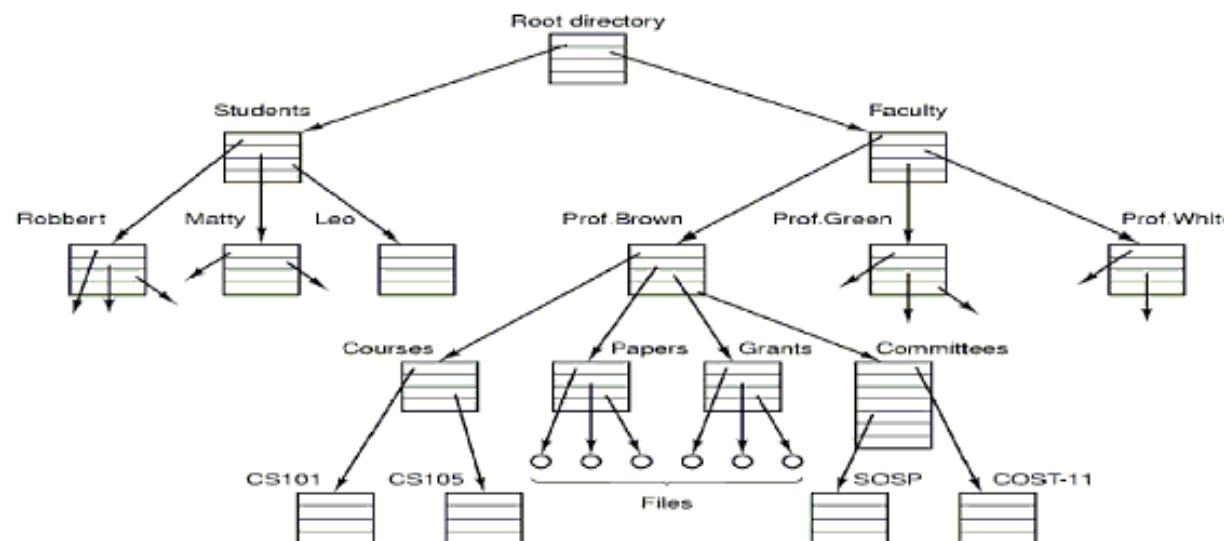
## Dateiattribute

- **Schutzinfos:** **wer kann wie zugreifen, Ersteller, Eigentümer**
- **Flags:** **Read-only, Hidden, System, Archive, Random access, Temporary**
- **Zeitfelder:** **Erstellung, letzter Zugriff, letzte Änderung**
- **Größe:** **aktuelle, maximale**

# 7.2 Verzeichnisse

## Verzeichnisse

- Strukturierung der Dateiablage
  - eine Ebene: nur Wurzelverzeichnis => keine gleiche Namen möglich
  - zwei Ebenen: Wurzel- und Benutzerverzeichnisse => keine gleiche Dateinamen für einen Benutzer
- => hierarchische Verzeichnissysteme:
- absolute/relative Pfadnamen
  - . ist aktuelles Verzeichnis
  - .. ist übergeordnetes Verzeichnis



# 7.3 Das UNIX-Dateisystem

Datei: Bytefolge, keine Struktur, Interpretation des Inhaltes obliegt der Anwendung/dem Besitzer  
255 Zeichen (ASCII) für Name

## Navigation in der Verzeichnishierarchie

\$ `pwd` - print working directory

\$ `cd [ dir ]` - change directory

\$ `ls [ dir1 [ dir2 [ ... ] ] ]` - Inhalt von Verzeichnissen anzeigen

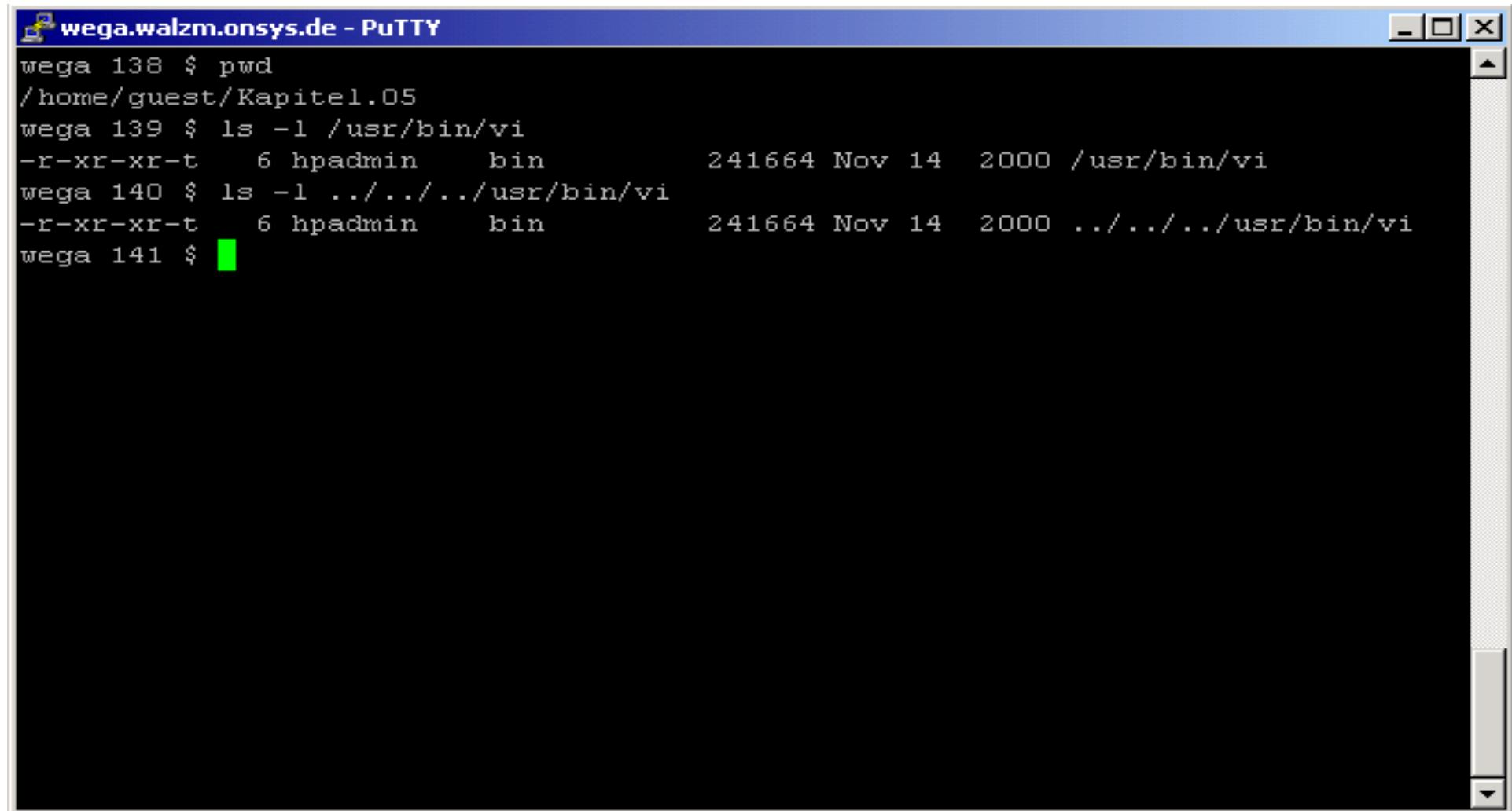
Option	Bedeutung
-l	ausführliches (long) Listing
-t	Zeitliche Ordnung der Ergebnisse
-a	Anzeige aller Dateien, auch der versteckten (hidden)
-d	nur Anzeige des Directorynamens, falls das Argument ein Verzeichnis ist
-R	rekursives Listing, d. h. Anzeige des Inhaltes aller Unterverzeichnisse

## Spezielle Verzeichnisse

Notation	Bedeutung
.	(Punkt) das Arbeitsverzeichnis
..(zwei Punkte)	Verzeichnis oberhalb des Arbeitsverzeichnisses
<code>~user</code>	Homeverzeichnis des Users <code>user</code> . Wenn <code>user</code> weggelassen wird, so ist das eigene Homeverzeichnis gemeint.
/	Wurzelverzeichnis (Rootdirectory)

# 7.3 Das UNIX-Dateisystem

## Absoluter und relativer Pfad

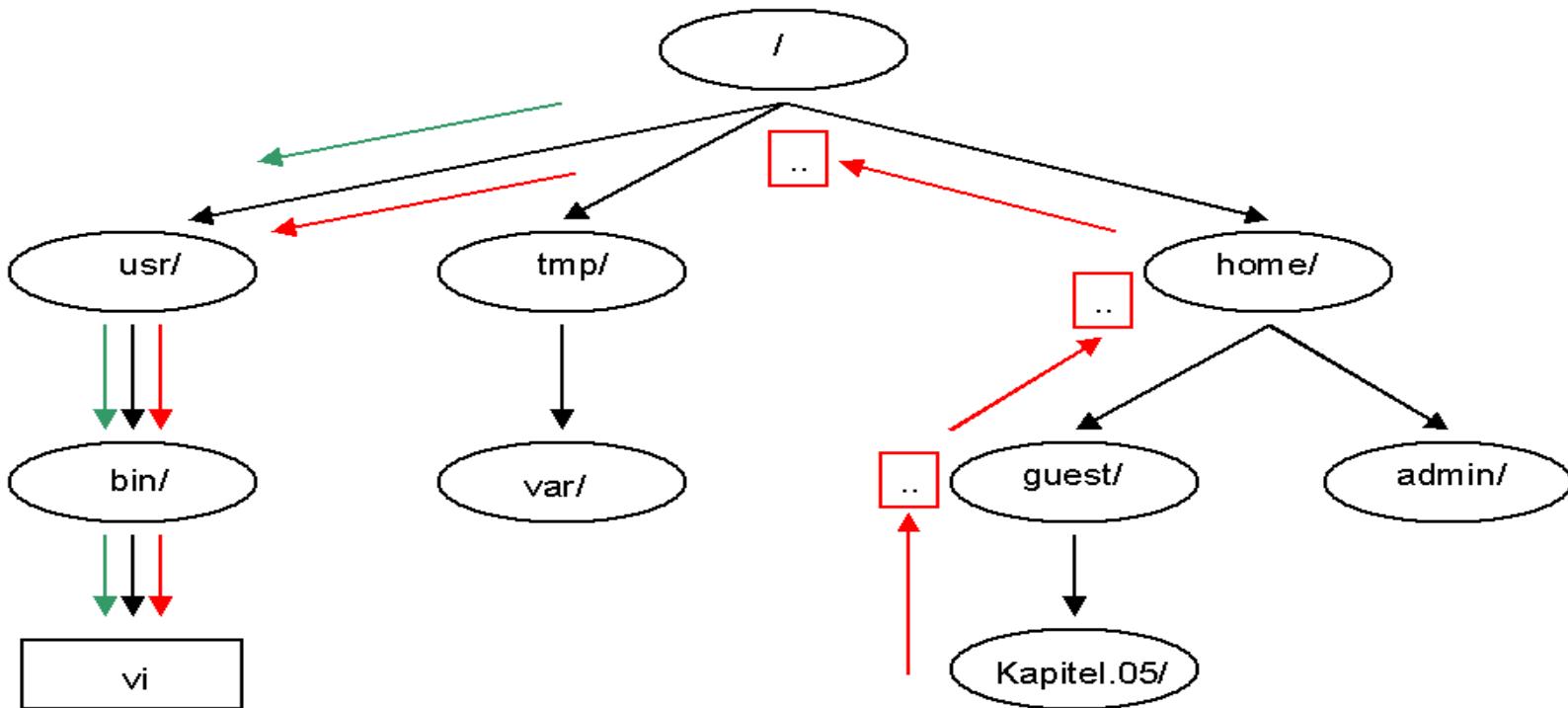


The screenshot shows a PuTTY terminal window titled "wega.walzm.onsys.de - PuTTY". The terminal displays the following command-line session:

```
wega 138 $ pwd  
/home/guest/Kapitel.05  
wega 139 $ ls -l /usr/bin/vi  
-r-xr-xr-t 6 hpadmin bin 241664 Nov 14 2000 /usr/bin/vi  
wega 140 $ ls -l ../../../../../usr/bin/vi  
-r-xr-xr-t 6 hpadmin bin 241664 Nov 14 2000 ../../../../../usr/bin/vi  
wega 141 $ █
```

The terminal window has a blue header bar with the title and standard window controls (minimize, maximize, close). The main area is black with white text. A vertical scroll bar is visible on the right side of the terminal window.

# 7.3 Das UNIX-Dateisystem



rot: relativer Pfad

grün: absoluter Pfad

# 7.3 Das UNIX-Dateisystem

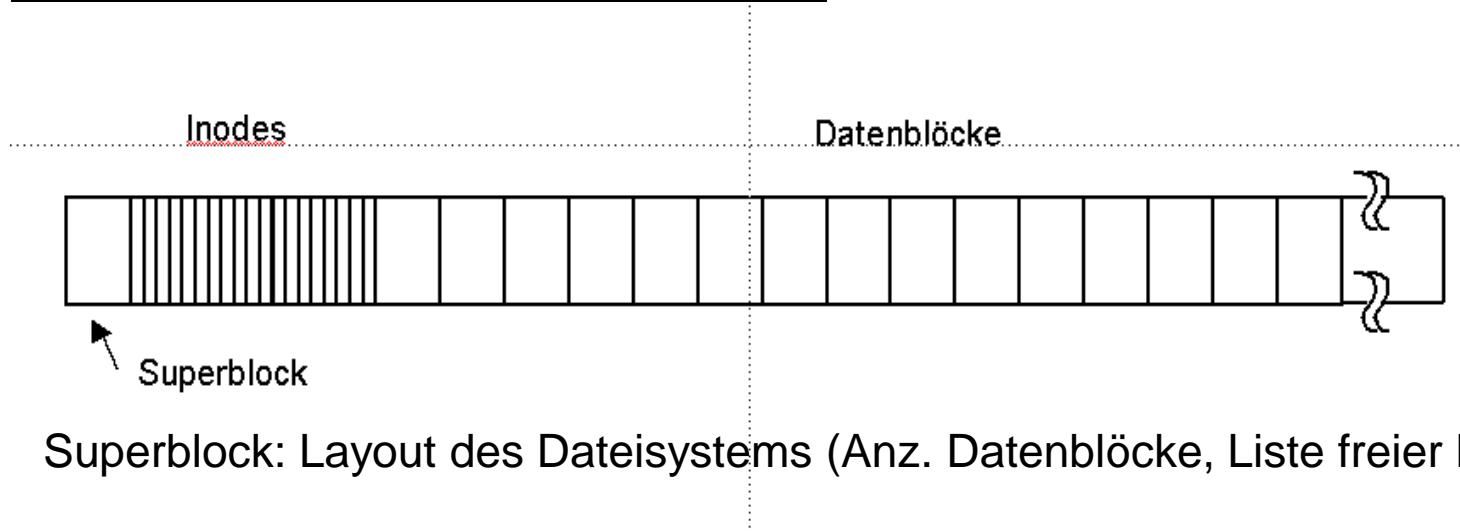
---

## Weitere Befehle

touch	Anlegen von Dateien
rm	Löschen von Dateien
mkdir [ -p ]	Anlegen von Directories
rmdir	Löschen von Verzeichnissen
rm -r	rekursives Löschen

# 7.3 Das UNIX-Dateisystem

## klassisches Dateisystem UNIX Version7



Superblock: Layout des Dateisystems (Anz. Datenblöcke, Liste freier Blöcke, Anz. Inodes)

I-Node: einer zu jeder Datei (dient dem Auffinden der Datei im Dateisystem) und enthält

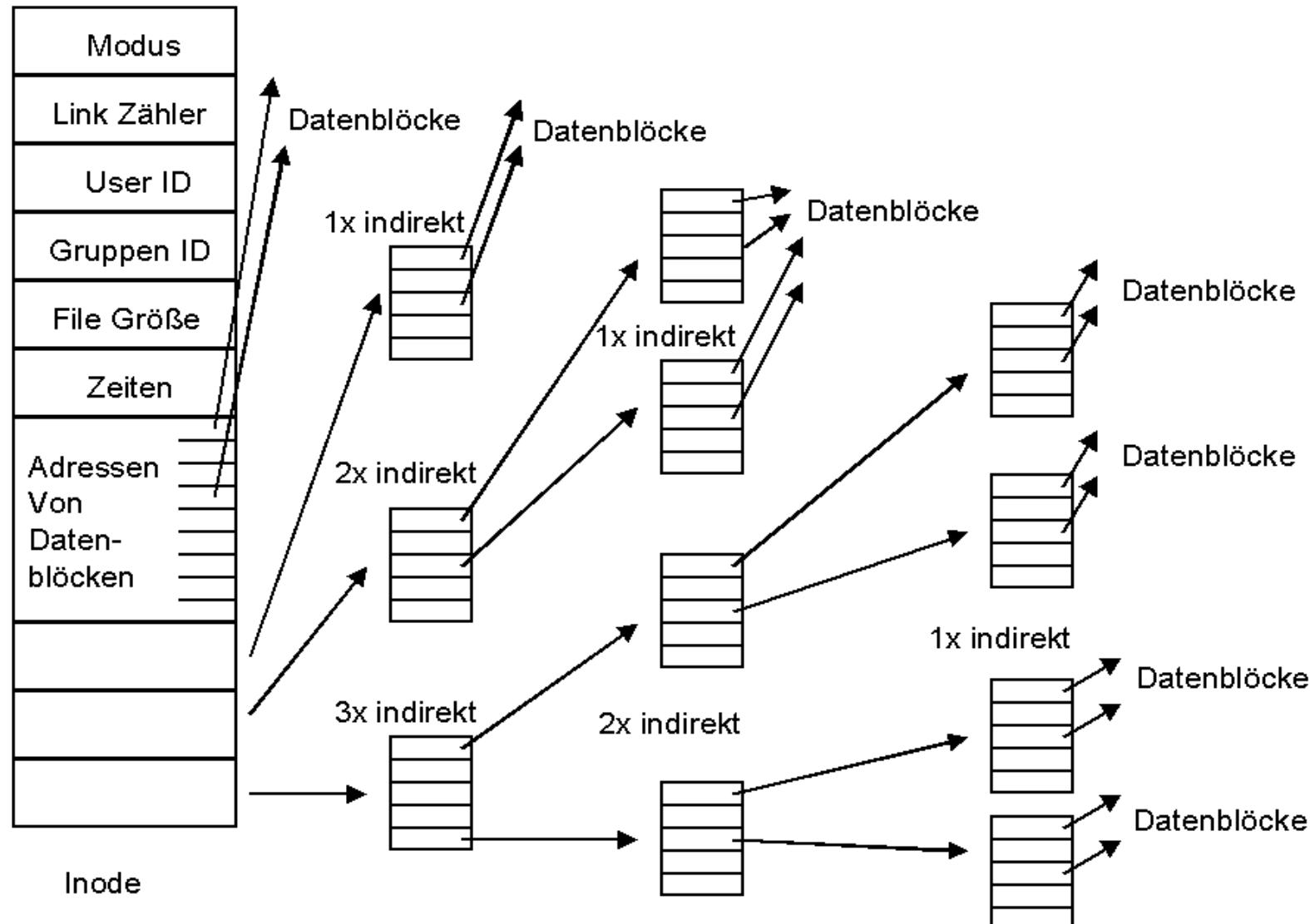
- Adressen der Datenblöcke
- Dateigröße
- drei Zeiten (Erzeugung, letzter Zugriff, letzte Änderung)
- Eigentümer, Gruppe
- Schutzinformationen
- Zähler, wie viele Verzeichniseinträge auf I-Node zeigen (bei Link Erhöhung)

Datenblöcke: Dateiinhalte

Directory: Tabelle mit zwei Spalten: I-Nodenummern und Dateinamen

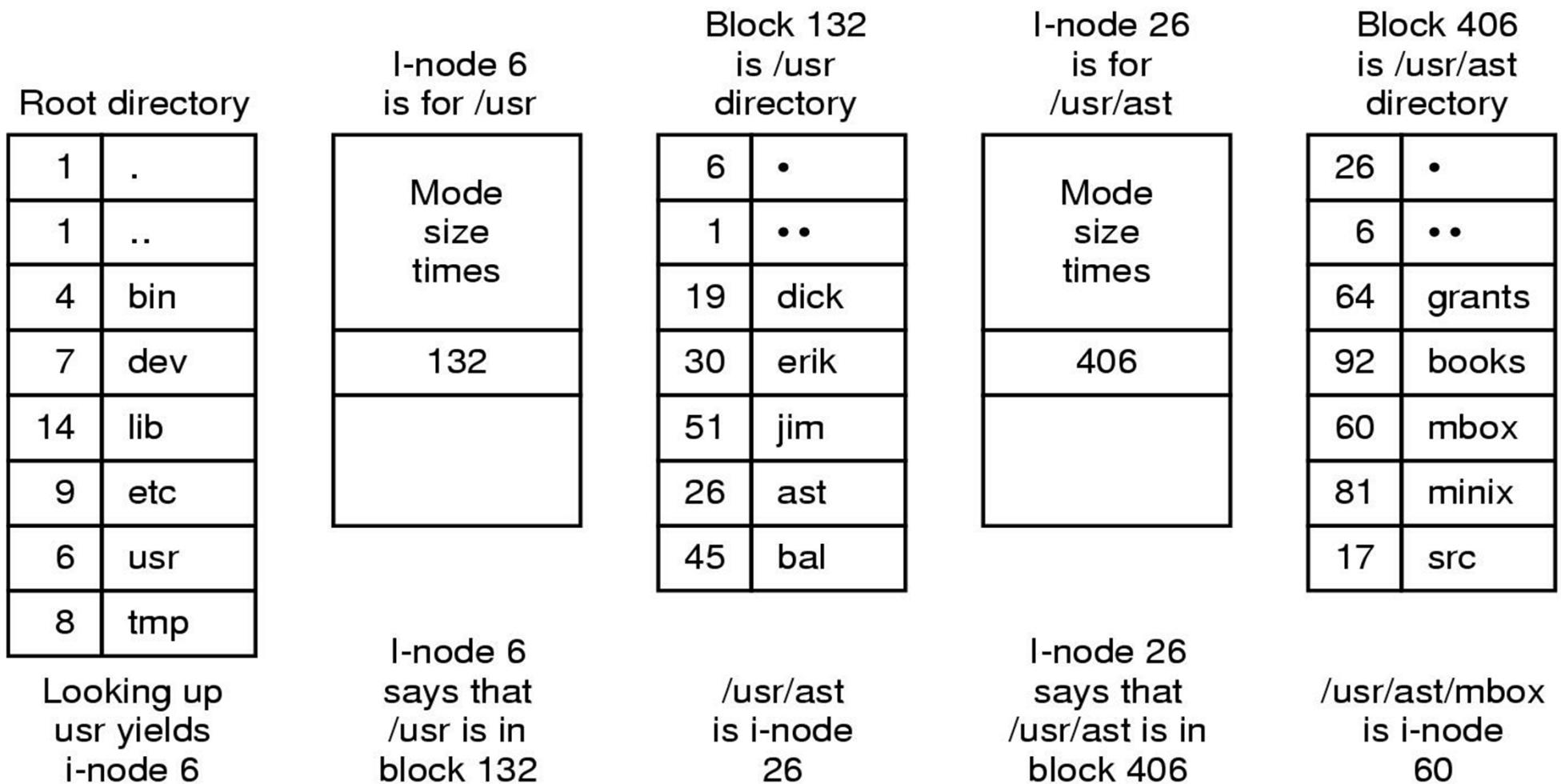
# 7.3 Das UNIX-Dateisystem

## Schematischer Aufbau eines I-Nodes



# 7.3 Das UNIX-Dateisystem

Auffinden der Plattenadressen: (Bsp.: Lokalisierung von /usr/ast/mbox)



# 7.3 Das UNIX-Dateisystem

## UNIX-Spezialdateien:

Der UNIX-Kernel kommuniziert mit der Peripherie durch Dateien.

Kennzeichnung	Dateityp	Erläuterung
- (Strich)	gewöhnliche Datei	In gewöhnlichen Dateien werden Daten logisch gespeichert.
d	Directory	UNIX behandelt Directories nur als eine spezielle Art von Dateien. Directories dienen der logischen Gruppierung von Dateien.
l	Link	Ein Link ist ein weiterer Name für eine andere Datei.
c	Character Device	Kommunikation mit zeichenorientierten Geräten (Terminal, Drucker, ...)
b	Block Device	Kommunikation mit blockorientierten Geräten (Festplatten, ...)
p	Named Pipe	Pipes sind Pseudodateien. Wir gehen später auf sie ein.
s	Domain Socket	Kommunikation von/mit Prozessen

Gerätedateien: major number (=Treiber), minor number (=Instanz des Gerätes)

Aufgabe:

Wechseln Sie ins Verzeichnis /dev und ermitteln Sie die den Typ, die Major und die Minor Number der Dateien hda, st1 und st0 mit dem Kommando ls und dem Kommando file!

# 7.3 Das UNIX-Dateisystem

---

## Übungen

1. Testen Sie die Optionen a, l, t und R des `ls`-Befehls.
2. Legen Sie unterhalb Ihres Homeverzeichnisses weitere Verzeichnisse an und löschen Sie diese wieder.

# 7.3 Das UNIX-Dateisystem

---

## Rechtestrukturen im Dateisystem:

### UID und GID

jeder Account durch diese zwei Werte charakterisiert (`/etc/passwd`)

Mitgliedschaft in mehreren Gruppen möglich (`/etc/group`)

### `/etc/passwd`

Username

verschlüsseltes Passwort (falls x, dann `/etc/shadow`)

UID

GID (primäre)

Kommentar

Homeverzeichnis

Loginshell

Superuser:

UID = 0, GID = 0

darf (fast) alles

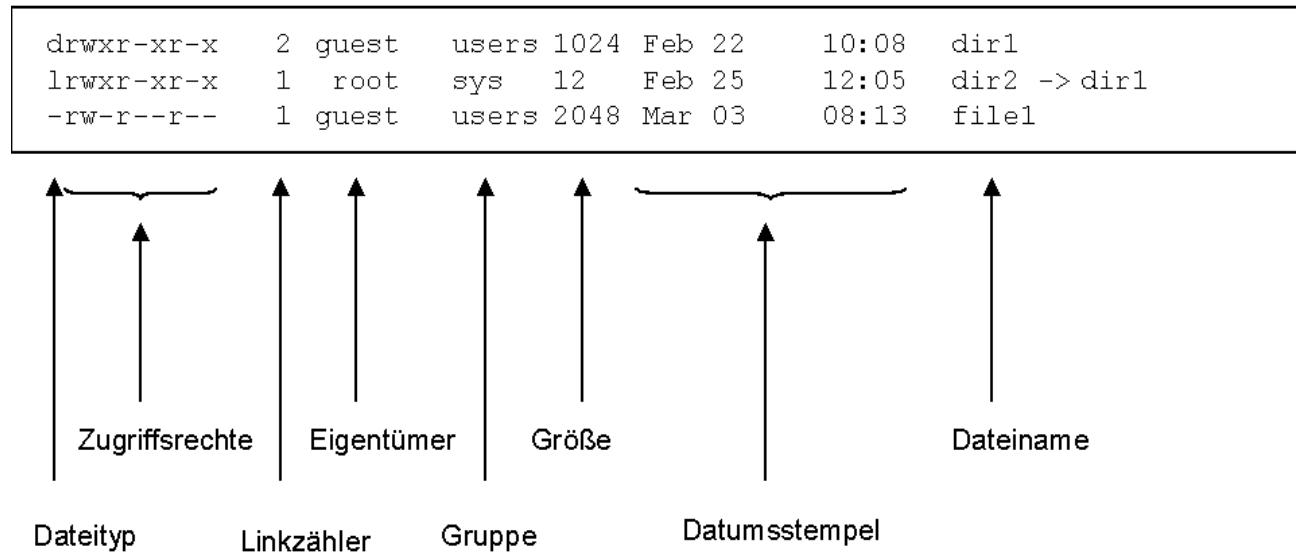
meist Name root

# 7.3 Das UNIX-Dateisystem

## Usergruppen und Rechtearten im Dateisystem

- read (r)
- write (w)
- execute (x)

- User (Eigentümer) (u)
- Group (g)
- Other (o)



- read: Befehle vi, cp, ...
- write: Befehl mv, ... (beim Löschen Schreibrecht des Verzeichnisses ausschlaggebend!)
- execute: run, bei Verzeichnissen Eintritt in Verzeichnis möglich mit cd

# 7.3 Das UNIX-Dateisystem

## Beeinflussung des Ausführungsverhaltens von Programmen

- Set-uid Bit: Prozess erhält für die Dauer der Ausführung Rechte des Dateieigentümers  
=> effektive User-ID (EUID)
- Set-gid-Bit: analog für Gruppenrechte
- Sticky Bit: heute meistens missachtet. Wenn auf Directories vergeben, darf jeder schreiben, aber nur Owner darf seine Objekte löschen (meist bei /tmp)

## Kodierungen des Rechteblocks

Symbolisch			Dual			Oktal			Bemerkung		
u	g	o	Ausf.	u	g	o	Ausf.	u	g	o	
r--	---	---	0	100	0	0	0	4	0	0	Leserecht Owner
-w-	---	---	0	10	0	0	0	2	0	0	Schreibrecht Owner
--x	---	---	0	1	0	0	0	1	0	0	Exec-Recht Owner
---	r--	---	0	0	100	0	0	0	4	0	Leserecht Gruppe
---	-w-	---	0	0	10	0	0	0	2	0	Schreibrecht Gruppe
---	--x	---	0	0	1	0	0	0	1	0	Exec-Recht Gruppe
---	---	r--	0	0	0	100	0	0	0	4	Leserecht Andere
---	---	-w-	0	0	0	10	0	0	0	2	Schreibrecht Andere
---	---	--x	0	0	0	1	0	0	0	1	Exec-Recht Andere
--s	---	---	100	0	0	0	4	0	0	0	set-uid Bit
---	--s	---	10	0	0	0	2	0	0	0	set-gid Bit
---	---	--t	1	0	0	0	1	0	0	0	Sticky Bit

# 7.3 Das UNIX-Dateisystem

---

## Typische Beispiele

<b>Symbolisch</b>	<b>Dual</b>	<b>Oktal</b>	<b>Erläuterung</b>
rwx -- --	000 111 000 000	0700	Eigentümer darf alles
rwx rwx --	000 111 111 000	0770	Eigentümer und Gruppe dürfen alles
r-- r- r-	000 110 100 100	0644	alle dürfen lesen, Eigentümer darf auch schreiben
rwx r-x r-x	000 111 101 101	0755	alle dürfen lesen und ausführen, Eigentümer darf auch schreiben

## Änderung des Eigentümers und der Zugriffsrechte

```
$ chown [ -R ] user:group file
$ chmod absolute_mode_list file
$ chmod relative_mode_list file
```

# 7.3 Das UNIX-Dateisystem

## Beispiele

Zugriffsrechte vorher			chmod-Parameter		Zugriffsrechte nachher		
u	g	o	symbolisch	absolut	u	g	o
rw-	rw-	rW-	+x	777	rwx	rwx	rwx
rwx	rwx	rwx	o-w	775	rwx	rwx	r-x
r-x	r-x	---	u+w	750	rwx	r-x	---
rwx	r-x	r--	g=rw	764	rwx	rw-	r--
rwx	r-x	r-x	=	0	---	---	---

# 7.3 Das UNIX-Dateisystem

---

## Übungen

1. Wem gehören die Dateien /bin/cp und /bin/mv?
2. Verändern Sie die Zugriffsrechte Ihres Homeverzeichnisses auf r-xr-xr-x.  
Versuchen Sie, eine beliebige Datei in dieses Verzeichnis  
a) zu kopieren, b) umzubenennen c) zu löschen.

# Zusammenfassung Kapitel 7

---

- Informationen, die persistent (dauerhaft) gespeichert werden, müssen unabhängig von der Entstehung und Beendigung eines Prozesses sein, und werden deshalb in Dateien gespeichert.
- Dateinamen genügen in verschiedenen Dateisystemen verschiedenen Namenskonventionen.
- Man unterscheidet verschiedene Dateitypen, wie reguläre Dateien (ASCII- oder Binärdateien), Verzeichnisse, spezielle Zeichendateien und spezielle Blockdateien.
- Der Zugriff auf eine Datei kann wahlfrei oder sequentiell erfolgen.
- Dateien haben verschiedene Attribute, wie Größe, Flags, Schutzinformationen und Zeitinformationen.
- Verzeichnisse ermöglichen eine strukturierte Ablage von Dateien. In der Praxis trifft man heute meist auf hierarchische Verzeichnissysteme.
- Das UNIX-Dateisystem ist hierarchisch aufgebaut und unterscheidet zwischen absolutem und relativem Pfad. Das Pfadtrennzeichen ist der Slash /
- Kommandos zur Navigation im Dateisystem sind pwd, cd und ls.
- Jede Datei wird durch einen I-Node beschrieben.
- UNIX unterscheidet verschiedene Zugriffsrechte auf Dateien.
- Kommandos zur Änderung von Zugriffsrechten sind chmod und chown.

# **Kapitel 8 - UNIX-Tools**

---

- 8.1 Operationen mit Dateien**
- 8.2 Operationen auf dem Dateisystem**
- 8.3 Zeichenkettenverarbeitung**
- 8.4 Kommunikation**
- 8.5 Terminaleinstellungen**

# 8.1 Operationen mit Dateien

---

## **cat      Ausgeben von Dateiinhalten (concatenate)**

Beispiele:

- |                   |   |
|-------------------|---|
| cat Datei1        | => gibt Inhalt von Datei1 aus                           |
| cat Datei1 Datei2 | => gibt Inhalt von Datei1 und Datei2 hintereinander aus |
| cat               | => liest von stdin, bis Strg+d                          |

## **more      Ausgeben von Dateiinhalten mit Stop nach einer Bildschirmseite**

Beispiel:

- |                  |                                     |
|------------------|-------------------------------------|
| more /etc/passwd | => Ausgabe von /etc/passwd mit Stop |
|------------------|-------------------------------------|

## **diff      Vergleichen von Textdateien (differences)**

Beispiel:

- |                                   |  |
|-----------------------------------|--|
| diff /etc/passwd passwd.yesterday | => gibt Unterschiede zwischen /etc/passwd und passwd.yesterday aus |
|-----------------------------------|--|

# 8.1 Operationen mit Dateien

---

## cp Kopieren von Dateien (copy)

Beispiele:

- |                  |  |
|------------------|--|
| cp Datei1 Datei2 | => erzeugt Kopie von Datei1 in Datei2  |
| cp -r Dir1 Dir2  | => wenn Dir2 nicht existiert, wird Verzeichnisbaum unter<br>Dir1 in Dir2 kopiert, sonst nach Dir2/Dir1 |

## mv Verschieben von Dateien (move)

Beispiele:

- |                  |                                     |
|------------------|-------------------------------------|
| mv Datei1 Datei2 | => Datei1 wird in Datei2 umbenannt  |
| mv Datei1 Dir1   | => Datei1 wird nach Dir1 verschoben |

## rm Löschen von Dateien (remove)

Beispiele:

- |            |  |
|------------|--|
| rm Datei1  | => löscht Datei1   |
| rm -r Dir1 | => löscht Verzeichnisbaum unterhalb Dir1 und Dir1 selbst |

# 8.1 Operationen mit Dateien

---

## **file      Bestimmen des Dateityps**

Beispiele:

file /dev/hda	=> /dev/hda: block special (3/0)
file /etc/passwd	=>/etc/passwd: ASCII text
file /usr/bin/passwd	=>/usr/bin/passwd: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), stripped

## **touch   Ändern des Zugriffszeitstempels für Dateien**

Beispiele:

touch NeueDatei	=> legt NeueDatei an
-----------------	----------------------

# 8.1 Operationen mit Dateien

---

## **g(un)zip**

### **(De)komprimierung von Dateien**

**Beispiele:**

gzip Datei1

=> Komprimierung in Datei1.gz

gunzip Datei1

=> Dekomprimierung von Datei1.gz nach Datei1

## **(un)compress**

### **(De)komprimierung von Dateien**

**Beispiele:**

compress Datei1

=> Komprimierung in Datei1.Z

uncompress Datei1.Z

=> Dekomprimierung in Datei1

=> gzip in Linux (Open Source), compress in UNIX (da kommerziell)

# 8.1 Operationen mit Dateien

---

## **head** Anfangszeilen einer Datei ausgeben

Beispiele:

head Datei1 => gibt die ersten 10 Zeilen von Datei1 aus

head -20 Datei1 => gibt die ersten 20 Zeilen von Datei1 aus

## **tail** Dateiinhalt ab bestimmter Position ausgeben

Beispiele:

tail Datei1 => gibt die letzten 10 Zeilen von Datei1 aus

tail -20 Datei1 => gibt die letzten 20 Zeilen von Datei1 aus

tail -f Datei1 => gibt in ständig wachsender Datei1 die neu hinzukommenden Zeilen aus

## **split** Zerteilen von Dateien

Beispiele:

split -l 10 Datei1 Teil => zerteilt Datei1 in Dateien Teil.aa, Teil.ab ... zu je 10 Zeilen

split -b 1024 Datei1 Teil => zerteilt Datei1 in 1KB große Dateien Teil.aa, Teil.ab ...

## 8.2 Operationen auf dem Dateisystem

---

**basename**      **Dateinamen aus Pfadnamen extrahieren**

Beispiele:

basename /home/guest/prog.c      => liefert prog.c

basename /home/guest/prog.c .c      => liefert prog

**dirname**      **Verzeichnisnamen aus Pfadnamen extrahieren**

Beispiele:

dirname /home/guest/prog.c      => liefert /home/guest

**find**      **Suchen von Dateien**

Beispiele:

find / -mtime 0 -size +1000000c      => Suche alle Dateien ab Verzeichnis /, die innerhalb der letzten 24 Stunden geändert wurden und die größer als 1 Million Bytes sind

find . -name core -exec rm {} \;      => Suche alle Dateien ab dem aktuellen Verzeichnis mit Namen core und lösche sie

=> sehr mächtiges Werkzeug

## 8.2 Operationen auf dem Dateisystem

### cpio

#### Erzeugen einer Archivdatei (copy in/out)

Beispiele:

find K7 -depth -print | cpio -o > K7.cpio => erzeugt Archiv mit allen Dateien unterhalb des Verzeichnisses K7 in einer Datei K7.cpio

cpio -i < K7.cpio => packt das Archiv im aktuellen Verzeichnis aus

cpio -ivt < K7.cpio => zeigt den Inhalt des Archivs an

find . -depth -print | cpio -pdmv /ziel => kopiert das aktuelle Verzeichnis mit allen Unterverzeichnissen unter Beibehaltung aller Rechte und Zeitstempel ins Verzeichnis /ziel

### tar

#### Erzeugen einer Archivdatei (tape archive)

Beispiele:

tar -cvf archiv.tar /home => alle Homeverzeichnisse in archiv.tar schreiben

tar -tvf archiv.tar => Inhalt von archiv.tar anzeigen

tar -xvf archiv.tar => Archiv im aktuellen Verzeichnis auspacken

# 8.2 Operationen auf dem Dateisystem

## **df** Ermitteln des freien Speicherplatzes (disk free)

## Beispiele:

`df` => Ausgabe der Filesystemdaten in 1K-Blöcken

`df -h` => Ausgabe der Filesystemdaten menschenlesbar

## **du      Ermitteln der Plattenplatzbelegung (disk usage)**

## Beispiele:

du => belegter Speicherplatz in jedem Verzeichnis ab dem  
aktuellen in 1K-Blöcken

du -h

du -s => belegter Speicherplatz ab aktuellem Verzeichnis

## 8.3 Zeichenkettenverarbeitung

---

**cut**                    **Spalten und Felder aus Dateien ausschneiden**

Beispiel:

`cat /etc/passwd | cut -d: -f1,6`                    => Feld 1 und 6 bzgl. Trennzeichen : aus /etc/passwd ausgeben

**grep**                    **Suchen in Dateien**

Beispiele:

`grep -i "m[ea][iy]er" /etc/passwd`                    => gibt alle Zeilen der /etc/passwd aus, die meier, meyer ,maier, mayer case-insensitiv enthalten

`ps -aef | grep sh`    => liefert alle Zeilen der Prozessliste, in denen sh vorkommt

`grep -v root /etc/passwd`                                    => liefert alle Zeilen der /etc/passwd, in denen nicht root vorkommt

# 8.3 Zeichenkettenverarbeitung

## wc      Zählen von Zeilen, Wörtern und Zeichen (word count)

Beispiele:

wc -l /etc/passwd	=> zählt die Zeilen in /etc/passwd = Useranzahl
wc -c /etc/passwd	=> zählt die Zeichen in /etc/passwd
wc -w /etc/passwd	=> zählt Wörter in /etc/passwd
wc /etc/passwd	=> zählt alle drei Arten
grep -v root /etc/passwd   wc -l	=> zählt alle Zeilen in /etc/passwd, in denen nicht root vorkommt

Übungen:

1. Lassen Sie sich die Zeile der /etc/passwd ausgeben, die Ihren Account beschreibt!  
Verwenden Sie dabei einmal direkt das Kommando und einmal eine Pipeline!
2. Lassen Sie sich alle anderen Zeilen der /etc/passwd ausgeben!
3. Schreiben Sie eine Pipeline, die Ihnen alle Umgebungsvariablen anzeigt, in denen Ihr Loginname vorkommt!
4. Wieviele Benutzer sind auf dem System eingerichtet?
5. Lassen Sie sich die ersten 5 Zeilen der /etc/passwd anzeigen!
6. Lassen Sie sich alle Zeilen ab der 17. Zeile der /etc/passwd ausgeben!

# 8.4 Kommunikation

---

## Mailclients:

elm, pine, mailx

## Datentransfer:

ftp	File Transfer Protocol
rcp	Remote Copy
scp	Secure copy (wie rcp, allerdings verschlüsselte Übertragung)
sftp	Secure FTP (wie ftp, aber verschlüsselt)

## Login:

telnet	Shell-Client
rlogin	Remote Login
ssh	Secure Shell (wie telnet, aber verschlüsselt)
who	wer ist am System angemeldet

# 8.5 Terminaleinstellungen

---

## stty

### Setzen und Abfragen von Terminaleinstellungen

Beispiele:

stty

=> Ausgeben der aktuellen Terminaleinstellungen

stty -echo

=> schaltet Echo der eingegebenen Zeichen ab (z.B. für  
Passwortabfrage)

stty echo

=> schaltet Echo wieder ein

stty sane

=> Wiederherstellung von Terminaleinstellungen bei  
Verstellen des Terminals

stty </dev/pts/1

=> Terminaleinstellungen des Terminals /dev/pts/1

## tty

### Erfragen des Terminalnamens

Beispiel:

tty

=> liefert z.B. /dev/pts/0

Übungen:

1. Überprüfen mit dem Mailclient mailx, ob neue Mail für Sie da ist!
2. Schalten sie das Echo Ihres Terminals aus und wieder ein!
3. Lassen Sie sich alle Benutzer anzeigen, die gerade am System angemeldet sind!