

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

Sommersemester 2025

## Wichtiger Hinweis

Diese Vortragsfolien dienen dazu, den Vortrag der/des Lehrenden zu unterstützen. Sie enthalten **nur einen Teil** der Lerninhalte. Wie groß dieser Teil ist, hängt von den konkreten Lerninhalten ab und kann von „praktisch alles“ bis „praktisch gar nichts“ schwanken. Diese Folien alleine sind daher **nicht für ein Selbststudium geeignet!**

Mindestens genauso wichtig wie die Vortragsfolien sind die Beispiel-Programme, Notizen und Tafelbilder, die vor Ihren Augen in den Vorlesungen erarbeitet werden. Diese sind im Git-Repository mit allen Zwischenschritten enthalten (<https://gitlab.cvh-server.de/pgerwinski/es>) und befinden sich in den zu den jeweiligen Kalenderdaten gehörenden Verzeichnissen (z. B. für den 24. 3. 2025 unter <https://gitlab.cvh-server.de/pgerwinski/bs/tree/2025ss/20250324/>).

In jedem Fall: *Viel Erfolg!*

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

24. März 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

24. März 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

24. März 2025

# Zu dieser Lehrveranstaltung

- Bitte nach Möglichkeit **eigenen Computer** (Notebook) mitbringen.
- **Lehrmaterialien:** <https://gitlab.cvh-server.de/pgerwinski/bs>  
Links auf die Datei klicken, nicht mittig auf den Kommentar.
- **Prüfungsform: Hausarbeit mit Kolloquium**

## Online-Teilnahme:

- **Mumble:** Seminarraum 2  
Fragen: Mikrofon einschalten oder über den Chat  
Umfragen: über den Chat
- **VNC:** Kanal 6, Passwort: `testcvh`  
Eigenen Bildschirm freigeben: per VNC-Server oder Web-Interface  
Kamerabild übertragen: Link zu Web-Interface auf Anfrage
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>  
Seminarraum 2, VNC-Passwort: `testcvh`

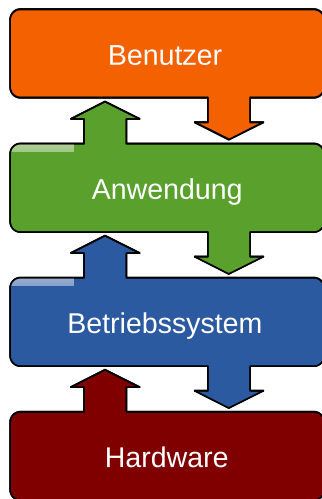
# 1 Einführung

Was ist ein Betriebssystem?

# 1 Einführung

Was ist ein Betriebssystem?

- Software, die zwischen Hardware und Anwendung vermittelt

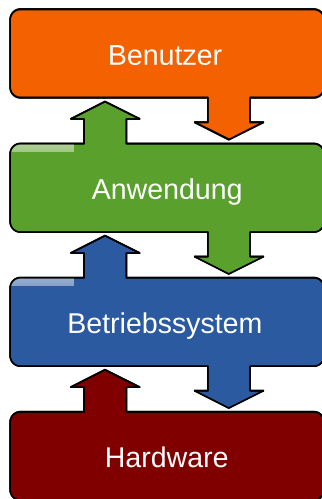




# 1 Einführung

Was ist ein Betriebssystem?

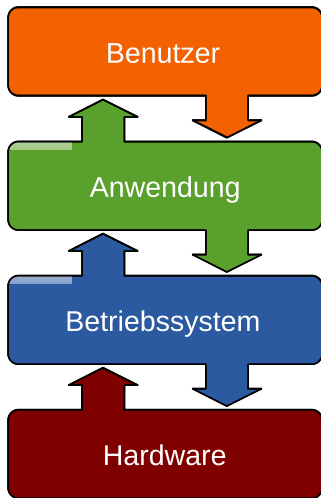
- Software, die zwischen Hardware und Anwendung vermittelt
- Mikro-Controller:  
Anwendung greift *direkt* auf Hardware zu



# 1 Einführung

Was ist ein Betriebssystem?

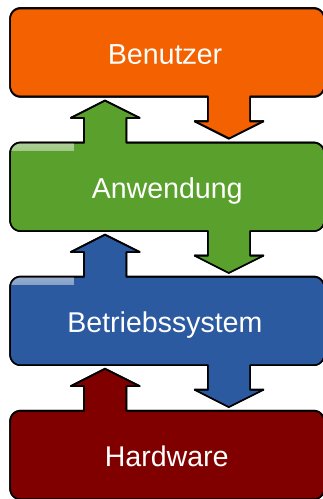
- Software, die zwischen Hardware und Anwendung vermittelt
- Mikro-Controller:  
Anwendung greift *direkt* auf Hardware zu
- Eingebettetes System:  
Anwendung startet automatisch
- Arbeitsplatz-Computer: *Oberfläche (Shell)*  
Benutzer\*in wählt Anwendung aus



# 1 Einführung

Was ist ein Betriebssystem?

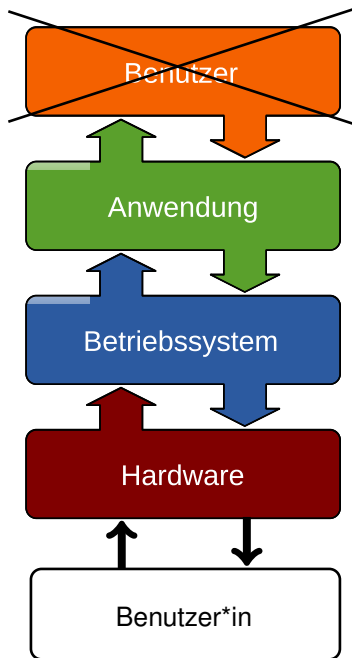
- Software, die zwischen Hardware und Anwendung vermittelt
- Mikro-Controller:  
Anwendung greift *direkt* auf Hardware zu
- Eingebettetes System:  
Anwendung startet automatisch
- Arbeitsplatz-Computer: *Oberfläche (Shell)*  
Benutzer\*in wählt Anwendung aus
- Ressourcen-Verwaltung



# 1 Einführung

Was ist ein Betriebssystem?

- Software, die zwischen Hardware und Anwendung vermittelt
- Mikro-Controller:  
Anwendung greift *direkt* auf Hardware zu
- Eingebettetes System:  
Anwendung startet automatisch
- Arbeitsplatz-Computer: *Oberfläche (Shell)*  
Benutzer\*in wählt Anwendung aus
- Ressourcen-Verwaltung



# 1 Einführung

Was gehört zum Betriebssystem?

- Betriebssystemkern: *Kernel*
- Benutzeroberfläche: *Shell*  
text- oder grafikorientiert  
(im engeren Sinne: Kommandozeile)
- Werkzeuge zur Verwaltung von Ressourcen  
(z. B. Festplatten formatieren)
- Graphische Benutzeroberfläche: *GUI*
- Texteditor
- Entwicklungswerkzeuge (Compiler usw.),  
Skriptsprachen
- Internet-Software: Web-Browser, E-Mail-Client usw.
- Multimedia-Software
- Büro-Anwendungssoftware

Ja, klar!

Hmm ... vielleicht.

# 1 Einführung

In dieser Lehrveranstaltung:

- Treiberentwicklung  
wie in *Hardwarenahe Programmierung* (3./5. Sem.), nur „größer“
- Echtzeitsysteme  
wie in *Eingebettete Systeme* (3./5. Sem.), nur „größer“
- neu: Betriebssysteme

# 1 Einführung

In dieser Lehrveranstaltung:

- Treiberentwicklung  
wie in *Hardwarenahe Programmierung* (3./5. Sem.), nur „größer“
- Echtzeitsysteme  
wie in *Eingebettete Systeme* (3./5. Sem.), nur „größer“
- neu: Betriebssysteme

Statt Klausur: Projektaufgabe, z. B.:

- neuartiger Treiber (z. B. für neuartige Hardware)
- neuartige Echtzeit-Funktionalität
- Sonstiges

# 1 Einführung

In dieser Lehrveranstaltung:

- Treiberentwicklung  
wie in *Hardwarenahe Programmierung* (3./5. Sem.), nur „größer“
- Echtzeitsysteme  
wie in *Eingebettete Systeme* (3./5. Sem.), nur „größer“
- neu: Betriebssysteme

Statt Klausur: Projektaufgabe, z. B.:

- neuartiger Treiber (z. B. für neuartige Hardware)
- neuartige Echtzeit-Funktionalität    **speziell:**
  - Echtzeitkommunikation  
für Home-Office und Online-Lehre
  - Treiber und Echtzeitkommunikation  
für freies Smartphone
- Sonstiges



# 1 Einführung

In dieser Lehrveranstaltung:

- Treiberentwicklung  
wie in *Hardwarenahe Programmierung* (3./5. Sem.), nur „größer“
- Echtzeitsysteme  
wie in *Eingebettete Systeme* (3./5. Sem.), nur „größer“
- neu: Betriebssysteme

Statt Klausur: Projektaufgabe, z. B.:

- neuartiger Treiber (z. B. für neuartige Hardware)
- neuartige Echtzeit-Funktionalität
- Sonstiges

speziell:

- Echtzeitkommunikation  
für Home-Office und Online-Lehre
- Treiber und Echtzeitkommunikation  
für freies Smartphone

Wiederholung:

- Unix
- Hardwarenahe Programmierung
- Theorie der Echtzeit-Systeme
- ? Sonstiges

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

31. März 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

31. März 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

31. März 2025

# Zu dieser Lehrveranstaltung

- Bitte nach Möglichkeit **eigenen Computer** (Notebook) mitbringen.
- **Lehrmaterialien:** <https://gitlab.cvh-server.de/pgerwinski/bs>  
Links auf die Datei klicken, nicht mittig auf den Kommentar.
- **Prüfungsform: Hausarbeit mit Kolloquium**

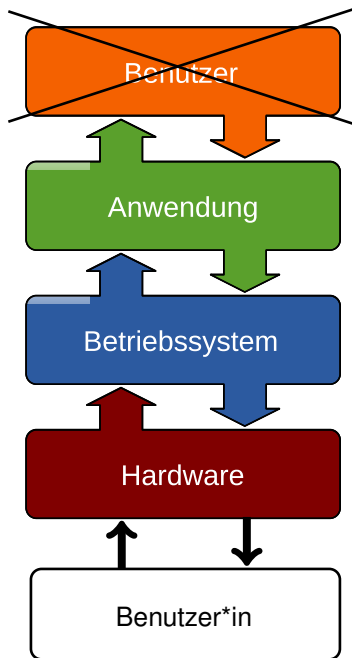
## Online-Teilnahme:

- **Mumble:** Seminarraum 2  
Fragen: Mikrofon einschalten oder über den Chat  
Umfragen: über den Chat
- **VNC:** Kanal 6, Passwort: `testcvh`  
Eigenen Bildschirm freigeben: per VNC-Server oder Web-Interface  
Kamerabild übertragen: Link zu Web-Interface auf Anfrage
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>  
Seminarraum 2, VNC-Passwort: `testcvh`

# 1 Einführung

Was ist ein Betriebssystem?

- Software, die zwischen Hardware und Anwendung vermittelt
- Mikro-Controller:  
Anwendung greift *direkt* auf Hardware zu
- Eingebettetes System:  
Anwendung startet automatisch
- Arbeitsplatz-Computer: *Oberfläche (Shell)*  
Benutzer\*in wählt Anwendung aus
- Ressourcen-Verwaltung



# 1 Einführung

Was gehört zum Betriebssystem?

- Betriebssystemkern: *Kernel*
- Benutzeroberfläche: *Shell*  
text- oder grafikorientiert  
(im engeren Sinne: Kommandozeile)
- Werkzeuge zur Verwaltung von Ressourcen  
(z. B. Festplatten formatieren)
- Graphische Benutzeroberfläche: *GUI*
- Texteditor
- Entwicklungswerkzeuge (Compiler usw.),  
Skriptsprachen
- Internet-Software: Web-Browser, E-Mail-Client usw.
- Multimedia-Software
- Büro-Anwendungssoftware

Ja, klar!

Hmm ... vielleicht.

# 1 Einführung

In dieser Lehrveranstaltung:

- Treiberentwicklung  
wie in *Hardwarenahe Programmierung* (3./5. Sem.), nur „größer“
- Echtzeitsysteme  
wie in *Eingebettete Systeme* (3./5. Sem.), nur „größer“
- neu: Betriebssysteme

Statt Klausur: Projektaufgabe, z. B.:

- neuartiger Treiber (z. B. für neuartige Hardware)
- neuartige Echtzeit-Funktionalität
- Sonstiges

speziell:

- Echtzeitkommunikation  
für Home-Office und Online-Lehre
- Treiber und Echtzeitkommunikation  
für freies Smartphone

Wiederholung:

- Unix
- Hardwarenahe Programmierung
- Theorie der Echtzeit-Systeme
- ? Sonstiges



## Praxisaufgabe: Aufbau eines grundlegenden TCP/IP-Netzwerks

Treiberentwicklung, Echtzeit- und Betriebssysteme · Sommersemester 2025 · Prof. Dr. Peter Gerwinski

Aufgabe: Vernetzen Sie Ihre Rechner mittels TCP/IP.

- Stellen Sie eine geeignete Hardware-Infrastruktur her.  
(Kabelgebundenes Netz, WLAN, Avian Carriers, ...)
- Wählen Sie geeignete IP-Adressen und prüfen Sie mittels `ping` die gegenseitige Erreichbarkeit.
- Stellen Sie mittels Netcat (`nc`) TCP-Verbindungen her (Ende-zu-Ende-Chat).
- Bieten Sie (z. B. mittels `nc -e`) in diesem Netz Dienste an und testen Sie diese.
- Untersuchen Sie den Netzwerkverkehr mittels `tcpdump` und/oder Wireshark.
- Testen Sie, wie sich externe Dienste (Webseiten, E-Mail) mittels `nc` nutzen lassen.

*Viel Erfolg!*

Stand: 10. April 2025

Copyright © 2025 Peter Gerwinski

Lizenz: CC BY-SA (Version 4.0) oder GNU GPL (Version 3 oder höher)

Sie können diese Praxisaufgabe einschließlich L<sup>A</sup>T<sub>E</sub>X-Quelltext herunterladen unter:

<https://gitlab.cvh-server.de/pgerwinski/ba>

## Versuchsskizze

1. Für eine Verbindung zwischen mehreren Geräten gibt es diverse Möglichkeiten:

- Ethernet (IEEE 802.3), bei typischen aktuellen PCs max. 1-10 Gbit/s möglich, bei Verbindung von zwei Geräten mittels Kabel (ab Gigabit ist hier kein Crossover-Kabel notwendig), bei mehr Geräten mit Hub oder Switch (im einfachsten Fall, sofern keine Segmentierung gewünscht). Unter Linux erscheinen die dann als Gerät *eth0*, *eth1*, ... oder *enp...* (für PCI-Geräte) oder *enx* (für Geräte mit USB-Anbindung), zur Benennung gibt es Details unter [https://systemd.io/PREDICTABLE\\_INTERFACE\\_NAMES/#what-precisely-has-changed-in-v197](https://systemd.io/PREDICTABLE_INTERFACE_NAMES/#what-precisely-has-changed-in-v197) und <https://www.freedesktop.org/software/systemd/man/latest/systemd.net-naming-scheme.html>
- Wireless-LAN (IEEE 802.11), theoretisch bei idealen Bedingungen Übertragung im Gigabit/s-Bereich möglich, aber ein geteiltes Medium und meist geringere Datenraten erreichbar. Hier wird typischerweise ein Access-Point zur Verbindung mehrerer Geräte benutzt, unter Linux erscheinen die Geräte z. B. als *wlan0* oder *wlp0s20i3*.
- Verbindung zweier Geräte per USB-C (USB3.2: maximal 20 Gbit/s, USB4: maximal 80 Gbit/s), erscheint unter Linux z. B. als *thunderbolt0*.
- Jeder Netzwerkadapter erhält vom Hersteller eine eindeutige Hardwareadresse, die normalerweise nicht verändert wird.

Im Praxisversuch wurden Ethernet und USB-C verwendet.

Damit Rechner (ohne Router oder über ein Point-to-Point-Protokoll) miteinander sprechen können, müssen sie sich im selben **Subnetz** befinden. Die Subnetzmaske bzw. Präfixlänge gibt an, wieviele Bits der IP-Adressen der Netzwerkteil sind, der Rest unterscheidet die einzelnen Hosts innerhalb eines Netzes, so ist z.B. bei der IPv4-Adresse 192.168.54.73/24 der Netzwerkteil 24 Bit lang und der Hostteil 32-24=8 Bit lang, die Hosts dürfen sich also nur in der letzten 8-Bit-Zahl unterscheiden. Zur Vergabe einer IP-Adresse gibt es mehrere Möglichkeiten:

- Manuelle Vergabe: Temporär z.B. mit `sudo ip addr add 198.51.100.1/30 dev enp0s31f6` oder dauerhaft mittels Network-Manager (entweder graphisch oder über

```
nmcli connection add con-name Beispielnetz type ethernet \
    ifname enp0s31f6 ipv4.addresses 198.51.100.1/30
```

- Vergabe per DHCP (insb. bei IPv4 verbreitet): Ein oder mehrere Server vergeben auf Anfrage IP-Adressen und informieren über die Adresse des Default-Gateways, typischerweise werden auch Zusatzinformationen, wie z.B. die Subnetzmaske, DNS-Server und die Gültigkeitsdauer der DHCP-Adresse mitgeliefert. Eine IPv4-Adresse kann z.B. temporär per `dhclient -eth0` oder dauerhaft per

```
nmcli connection add con-name Beispielnetz type ethernet \
    ifname enp0s31f6 ipv4.method auto ipv4.local disabled
```

(mit explizit deaktiviertem Fallback auf *link-lokal*-Adressen) konfiguriert werden.

Automatische Auswahl einer lokalen Adresse (bei IPv6 üblich, bei IPv4 eher als Fallback, falls kein DHCP-Server erreichbar ist). Relevant sind hierfür insb. **Dynamic Configuration of IPv4 Link-Local Addresses** bzw. **SLAAC**. Der Rechner wählt selbstständig eine Adresse aus dem Bereich 169.254.0.0/16 (IPv4) bzw. ff02::1/16 (IPv6) aus und überprüft mittels Address Resolution Protocol (IPv4), ob diese noch nicht verwendet wird (und wählt ggf. eine andere Adresse), bei IPv6 wird diese von der Hardware-Adresse abgeleitet und ist dadurch automatisch eindeutig. Explizite Konfiguration unter IPv4 z. B. mittels

```
nmcli connection add con-name Beispielnetz type ethernet \
    ifname enp0s31f6 ipv4.method link-local
```

bei IPv6 wird eine lokale Adresse zwingend benötigt.

- Eigenständige Auswahl einer globalen IPv6-Adresse mittels SLAAC und des **Neighbor Discovery Protocol**: Übliche Vorgehensweise bei IPv6, hierfür müssen im Netzwerk entsprechende **Router Advertisements** verteilt werden, unter Linux mittels `radvd`.

In Praxisversuch wurde zur Kommunikation vor Ort IPv4 mit link-local-Adresse verwendet.

Für die Anbindung der Systeme außerhalb der Hochschule wurde als Overlay-Netz **The onion router** verwendet. Unter Debian-basierten Linux-Distributionen ist die Installation mittels `apt install tor` möglich. Um hierüber eine Kommunikationsverbindung herzustellen, werden zwei TOR-Instanzen benötigt:

- Zunächst ist auf Serverseite die Konfigurationsdatei `/etc/tor/torrc` anzupassen (ggf. mit root-Rechten):

```
HiddenServiceDir /var/lib/tor/beispieldienst
HiddenServicePort 1234 127.0.0.1:22
```

Hiermit wird im TOR-Netzwerk ein sog. Hidden-Service auf dem Port 1234 geöffnet (dieser ist hidden, weil von außen nicht ohne weiteres feststellbar ist, auf welchem Gerät dieser Dienst gehostet wird. Sobald der Server mit `systemctl restart tor` neu gestartet wird, wird dafür ein Schlüsselpaar erzeugt, der daraus abgeleitete Hostname ist dann unter `/var/lib/tor/beispieldienst/hostname` abgespeichert.

- Auf dem Clientsystem kann nun mittels des SOCKS-Proxy-Protokolls eine Verbindung zu diesem Dienst aufgebaut werden, z.B. mittels  
`torsocks ssh -p 1234 benutzername@3u9wrgiv7...hvoqoqbubg52fflwpeyd.onion`  
(ggf. muss torsocks zuvor installiert werden).

2. Die per Kabel verbundenen Rechner konnten sich untereinander per `ping`-Kommando kontaktieren, wobei ein Rechner auf Grund einer aktiven Firewall zunächst nicht antwortete. Mittels `ip neighbour` und `wireshark` konnte aber dennoch überprüft werden, dass dieser Rechner tatsächlich auf `arp`-Requests antwortete (per IPv6 wäre stattdessen NDP benutzt worden). Auch lies sich per `wireshark` beobachten, wie mittels `arp` überprüft wurde, ob die gewählte IPv4-Adresse noch frei war.

3. Mittels `nc -l -p 1234` wurde ein Chat-Dienst bereit gestellt und mittels `nc 169.254.4.2 1234` wurde eine Verbindung zu diesem aufgebaut.

Wenn in der `torrc` die Zeile, die mit `HiddenServicePort` beginnt zu

```
HiddenServicePort 1234 127.0.0.1:1234
```

geändert wird (und TOR neugestartet wird), kann im Anschluss auch mittels

```
torsocks nc 3u9wrgiv7...hvoqoqbubg52fflwpeyd.onion 1234
```

 ein Zugriff per TOR auf den Chatserver gestartet werden (je nach Netcat-Version wird ggf. auch von diesem selbst ein Zugriff per SOCKS über den TOR-Server auf `localhost:9050`).

4. Als Serverdienst wurde ein SSH-Server eingesetzt, über `ssh 169.254.4.2` konnte ein Verbindungsaufbau durchgeführt werden, alternativ ist auch mittels `nc 169.254.4.2 22` eine Verbindung aufgebaut worden, welche mit einer Meldung der Art `SSH-2.0-OpenSSH_9.9p2 Debian-2` beantwortet wurde und fortgeführt wurde, wenn über den Netcat-Client eine entsprechend aufgebaute Nachricht übermittelt wurde (getestet durch Copy-and-Paste der initialen Nachricht).

5. Geeignete Dienste für einen Test wären **HTTP** oder **SMTP**, entsprechende Dialoge sehen folgendermaßen aus:

Webseitenabruf per HTTP:

```
nc -C www.cvh-server.de 80
GET / HTTP/1.1
Host: www.cvh-server.de
```

```
HTTP/1.1 302 Found
Date: Thu, 10 Apr 2025 08:51:22 GMT
Server: Apache/2.4.62 (Debian)
Strict-Transport-Security: max-age=15768000; includeSubDomains
Location: https://www.cvh-server.de/
Content-Length: 293
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD_HTML_2.0//EN">
<html><head>
<title>302 Found</title>
```

```
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://www.cvh-server.de">here</a></p>
<hr>
<address>Apache/2.4.62 (Debian) Server at www.cvh-server.de Port 80</address>
</body></html>
```

E-Mailversand per (E)SMTP (Hinweis: Auch wenn Sie hier früher quasi beliebige Absenderdaten eintragen konnten, überprüfen einige Mailserver inzwischen, ob Ihre Absenderdaten zu Ihrem E-Mailaccount passen, auch gibt es Verfahren wie **DKIM**, **SPF** und **DMARC** mit denen überprüft werden kann, ob ein Mailserver berechtigt ist, Mails für eine bestimmte Domain zu erzeugen bzw. weiterzuleiten.)

```
$ nc -C localhost 25
220 rechnername ESMTP Exim 4.98.2 Thu, 10 Apr 2025 11:06:28 +0200
EHLO localhost
250-rechnername Hello localhost [:-1]
250-SIZE 52428800
250-LIMITS MAILMAX=1000 RCPTMAX=50000
250-8BITMIME
250-PIPELINING
250-PIPECONNECT
250-CHUNKING
250-STARTTLS
250-PRDR
250-HELP
MAIL FROM: praxisversuch@localhost.cvh-server.de
250 OK
RCPT TO: bwildenhain@cvh-server.de
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
Subject: Bitte schicken Sie uns Ihre PIN zu
From: support@badbank.example.org
To: bwildenhain@cvh-server.de

Bitte Senden Sie uns umgehendend Ihre PIN fuer Ihr Online-Banking zu um Ihr Konto zu verifizieren.
.
250 OK id=1u2nsR-000000006Ck-3mZP
quit
221 rechnername closing connection
```

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

28. April 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

## 1 Einführung

- 1.1 Was ist ein Betriebssystem?
- 1.2 Zu dieser Lehrveranstaltung

## 2 Unix

- 2.1 Grundkonzepte
- 2.2 Die Kommandozeile: Grundlagen
- 2.3 Dateisysteme
- 2.4 Ein- und Ausgabeströme
- 2.5 Pipes
- 2.6 Verzweigungen und Schleifen

## 3 Treiberentwicklung

- 3.0 Massenspeicher und Gerätedateien
- 3.1 Mikrocontroller
- 3.2 Betriebssysteme ohne Speicherschutz
- 3.3 Linux-Kernel-Module

...

## 3 Treiberentwicklung

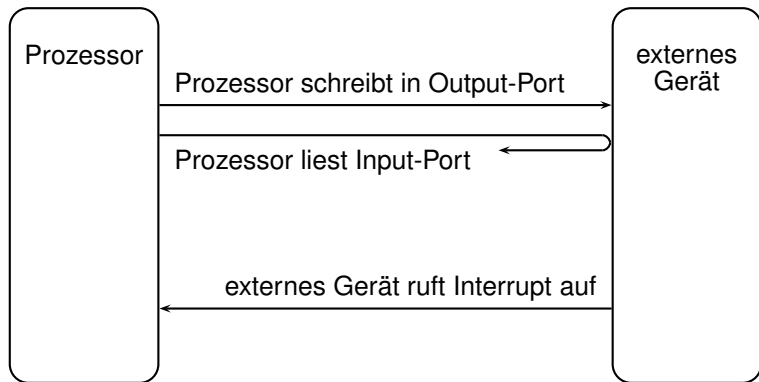
### 3.0 Massenspeicher und Gerätedateien

- Linux: Ein angeschlossener USB-Stick erscheint als Gerätedatei: `/dev/sdb`
- „Normale“ Benutzung: `mount`  
(Siehe: Unix, Datenträger *einhängen*)
- Gerätedatei: Direktzugriff auf die Bytes des Datenträgers  
Anwendungen: Datensicherung, Datenrettung

## 3 Treiberentwicklung

### 3.1 Mikrocontroller

Kommunikation mit externen Geräten





## 3.1 Mikrocontroller

### 3.1.1 I/O-Ports

In Output-Port schreiben = Aktoren ansteuern

Beispiel: LED

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0x70;    binär: 0111 0000
```

```
PORTC = 0x40;   binär: 0100 0000
```

Herstellerspezifisch!

DDR = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*

## 3.1 Mikrocontroller

### 3.1.1 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0xfd;          binär: 1111 1101
```

```
while ((PINC & 0x02) == 0) binär: 0000 0010
```

```
    ; /* just wait */
```

Herstellerspezifisch!

DDR = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*

Praktikumsaufgabe in *Hardwarenahe Programmierung*: Druckknopfampel

## 3.1 Mikrocontroller

### 3.1.2 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: eingebaute Uhr

statt Zählschleife (`_delay_ms`):  
Hauptprogramm kann  
andere Dinge tun

```
#include <avr/interrupt.h>
```

... „Dies ist ein Interrupt-Handler.“

Interrupt-Vektor darauf zeigen lassen

```
ISR (TIMER0B_COMP_vect)
{
    PORTD ^= 0x40;
}
```

Herstellerspezifisch!

Initialisierung über spezielle Ports: `TCCR0B`, `TIMSK0`

*Details: siehe Datenblatt und Schaltplan*

## 3.1 Mikrocontroller

### 3.1.2 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
ISR (INT0_vect)
```

```
{  
    PORTD ^= 0x40;  
}
```

statt *Busy Waiting*:  
Hauptprogramm kann  
andere Dinge tun

Herstellerspezifisch!

Initialisierung über spezielle Ports: `EICRA`, `EIMSK`

*Details: siehe Datenblatt und Schaltplan*

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{
```

```
    key_pressed = 1;
```

```
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
    PORTD ^= 0x40;
```

```
    key_pressed = 0;
```

```
}
```

```
return 0;
```

```
}
```

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
volatile uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{  
    key_pressed = 1;  
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
    PORTD ^= 0x40;
```


```
    key_pressed = 0;
```

```
}
```

```
return 0;
```

```
}
```

**volatile:**  
Speicherzugriff  
nicht wegoptimieren



## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```



## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (*(volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (* (volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

Zahl: 0x2B

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

→  $(*(\underbrace{\text{volatile uint8\_t } *}_{\text{Umwandlung in Zeiger auf volatile uint8\_t}})(\underbrace{(0x0B) + 0x20}_{\text{Zahl: 0x2B}})) = 0x01;$

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (* (volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

Umwandlung in Zeiger  
auf **volatile** uint8\_t

Zahl: 0x2B

Dereferenzierung des Zeigers

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (* (volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

Umwandlung in Zeiger  
auf **volatile** uint8\_t

Zahl: 0x2B

Dereferenzierung des Zeigers

→ **volatile** uint8\_t-Variable an Speicheradresse 0x2B

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (* (volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

Umwandlung in Zeiger  
auf **volatile** uint8\_t

Zahl: 0x2B

Dereferenzierung des Zeigers

→ **volatile** uint8\_t-Variable an Speicheradresse 0x2B

→ PORTA = PORTB = PORTC = PORTD = 0 ist eine schlechte Idee.

## 3.2 Betriebssysteme ohne Speicherschutz

- FreeDOS (früher: PC-DOS, MS-DOS, DR-DOS, Novell-DOS, aber auch Apple-DOS, Commodore-Kernal, ...):

Direkter Zugriff auf Speicher, I/O-Ports und Interrupts

## 3.2 Betriebssysteme ohne Speicherschutz

- FreeDOS (früher: PC-DOS, MS-DOS, DR-DOS, Novell-DOS, aber auch Apple-DOS, Commodore-Kernal, ...):  
Direkter Zugriff auf Speicher, I/O-Ports und Interrupts
- Bildschirm (Textmodus): ab Speicherzelle **B800:0000**  
abwechselnd Zeichen (CP 437) und Attribut (Farbe, Hintergrund)



## 3.2 Betriebssysteme ohne Speicherschutz

- FreeDOS (früher: PC-DOS, MS-DOS, DR-DOS, Novell-DOS, aber auch Apple-DOS, Commodore-Kernal, ...):  
Direkter Zugriff auf Speicher, I/O-Ports und Interrupts
- Bildschirm (Textmodus): ab Speicherzelle **B800:0000**  
abwechselnd Zeichen (CP 437) und Attribut (Farbe, Hintergrund)
- Bildschirm (Grafikmodus): später

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

5. Mai 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

## 1 Einführung

## 2 Unix

## 3 Treiberentwicklung

### 3.0 Massenspeicher und Gerätedateien

### 3.1 Mikrocontroller

### 3.2 Betriebssysteme ohne Speicherschutz

### 3.3 Linux-Kernel-Module

### 3.4 Die Systembibliothek (libc)

### 3.5 Der Betriebssystemkern (Kernel)

...

## 3 Treiberentwicklung

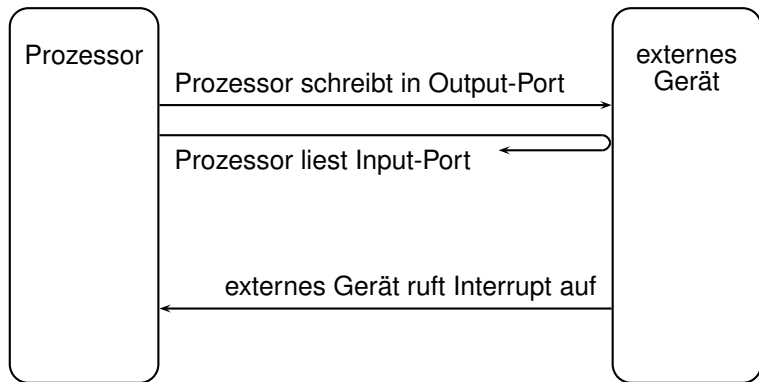
### 3.0 Massenspeicher und Gerätedateien

- Linux: Ein angeschlossener USB-Stick erscheint als Gerätedatei: `/dev/sdb`
- „Normale“ Benutzung: `mount`  
(Siehe: Unix, Datenträger *einhängen*)
- Gerätedatei: Direktzugriff auf die Bytes des Datenträgers  
Anwendungen: Datensicherung, Datenrettung

## 3 Treiberentwicklung

### 3.1 Mikrocontroller

Kommunikation mit externen Geräten



## 3.1 Mikrocontroller

### 3.1.1 I/O-Ports

In Output-Port schreiben = Aktoren ansteuern

Beispiel: LED

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0x70;    binär: 0111 0000
```

```
PORTC = 0x40;    binär: 0100 0000
```

Herstellerspezifisch!

DDR = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*



## 3.1 Mikrocontroller

### 3.1.1 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0xfd;          binär: 1111 1101
```

```
while ((PINC & 0x02) == 0) binär: 0000 0010
```

```
    ; /* just wait */
```

Herstellerspezifisch!

DDR = Data Direction Register

Bit = 1 für Output-Port

Bit = 0 für Input-Port

*Details: siehe Datenblatt und Schaltplan*

Praktikumsaufgabe in *Hardwarenahe Programmierung*: Druckknopfampel

## 3.1 Mikrocontroller

### 3.1.2 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: eingebaute Uhr

statt Zählschleife (`_delay_ms`):  
Hauptprogramm kann  
andere Dinge tun

```
#include <avr/interrupt.h>
```

... „Dies ist ein Interrupt-Handler.“

Interrupt-Vektor darauf zeigen lassen

```
ISR (TIMER0B_COMP_vect)
{
    PORTD ^= 0x40;
}
```

Herstellerspezifisch!

Initialisierung über spezielle Ports: `TCCR0B`, `TIMSK0`

*Details: siehe Datenblatt und Schaltplan*

## 3.1 Mikrocontroller

### 3.1.2 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
ISR (INT0_vect)
```

```
{  
    PORTD ^= 0x40;  
}
```

statt *Busy Waiting*:  
Hauptprogramm kann  
andere Dinge tun

Herstellerspezifisch!

Initialisierung über spezielle Ports: `EICRA`, `EIMSK`

*Details: siehe Datenblatt und Schaltplan*

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{
```

```
    key_pressed = 1;
```

```
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
    PORTD ^= 0x40;
```

```
    key_pressed = 0;
```

```
}
```

```
return 0;
```

```
}
```

## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Externes Gerät ruft (per Stromsignal) Unterprogramm auf  
Zeiger hinterlegen: „Interrupt-Vektor“

Beispiel: Taster

```
#include <avr/interrupt.h>
```

```
...
```

```
volatile uint8_t key_pressed = 0;
```

```
ISR (INT0_vect)
```

```
{  
    key_pressed = 1;  
}
```

```
int main (void)
```

```
{
```

```
...
```

```
while (1)
```

```
{
```

```
    while (!key_pressed)
```

```
        ; /* just wait */
```

```
    PORTD ^= 0x40;
```


```
    key_pressed = 0;
```

```
}
```

```
return 0;
```

```
}
```

**volatile:**  
Speicherzugriff  
nicht wegoptimieren



## 3.1 Mikrocontroller

### 3.1.3 volatile-Variable

Was ist eigentlich PORTD?

```
avr-gcc -Wall -Os -mmcu=atmega328p blink-3.c -E
```

```
PORTD = 0x01;
```

```
→ (* (volatile uint8_t *) ((0x0B) + 0x20)) = 0x01;
```

Umwandlung in Zeiger  
auf **volatile** uint8\_t

Zahl: 0x2B

Dereferenzierung des Zeigers

→ **volatile** uint8\_t-Variable an Speicheradresse 0x2B

→ PORTA = PORTB = PORTC = PORTD = 0 ist eine schlechte Idee.

## 3.2 Betriebssysteme ohne Speicherschutz

- FreeDOS (früher: PC-DOS, MS-DOS, DR-DOS, Novell-DOS, aber auch Apple-DOS, Commodore-Kernal, ...):  
Direkter Zugriff auf Speicher, I/O-Ports und Interrupts
- Bildschirm (Textmodus): ab Speicherzelle **B800:0000**  
abwechselnd Zeichen (CP 437) und Attribut (Farbe, Hintergrund)
- Bildschirm (Grafikmodus): später

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**



## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: `MODULE_LICENSE()`
- Kommunikation mit Anwendungen:  
Geräte-Dateien (*major/minor number*)

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**
- Kommunikation mit Anwendungen:  
Geräte-Dateien (*major/minor number*)
- Kommunikation mit dem Betriebssystemkern:  
Callback-Funktionen (virtuelle Methoden)

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**
- Kommunikation mit Anwendungen:  
Geräte-Dateien (*major/minor number*)
- Kommunikation mit dem Betriebssystemkern:  
Callback-Funktionen (virtuelle Methoden)
- Automatisches Anlegen von Gerätedateien:  
Geräteklassen

## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung
- Verbindung zum Betriebssystemkern

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

12. Mai 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

## 1 Einführung

## 2 Unix

## 3 Treiberentwicklung

3.0 Massenspeicher und Gerätedateien

3.1 Mikrocontroller

3.2 Betriebssysteme ohne Speicherschutz

3.3 Linux-Kernel-Module

3.4 Die Systembibliothek (libc)

3.5 Der Betriebssystemkern (Kernel)

...

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**
- Kommunikation mit Anwendungen:  
Geräte-Dateien (*major/minor number*)
- Kommunikation mit dem Betriebssystemkern:  
Callback-Funktionen (virtuelle Methoden)
- Automatisches Anlegen von Gerätedateien:  
Geräteklassen

## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung
- Verbindung zum Betriebssystemkern



## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf
- Verbindung zum Betriebssystemkern

## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf  
Indirekter Funktionsaufruf über eine Tabelle von Zeigern,  
abhängig von der Datei  
—→ Aufruf einer virtuellen Methode (objektorientiert)
- Verbindung zum Betriebssystemkern

## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf  
Indirekter Funktionsaufruf über eine Tabelle von Zeigern,  
abhängig von der Datei  
—→ Aufruf einer virtuellen Methode (objektorientiert)
- Verbindung zum Betriebssystemkern: System Call

## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf  
Indirekter Funktionsaufruf über eine Tabelle von Zeigern,  
abhängig von der Datei  
—→ Aufruf einer virtuellen Methode (objektorientiert)
- Verbindung zum Betriebssystemkern: System Call  
Das Betriebssystem hat mehr Rechte als die Anwendung,  
daher darf die Anwendung Funktionen des Betriebssystems  
nur auf sehr kontrollierte Weise aufrufen.  
—→ Hardware-Unterstützung  
Beispiel: x64-Befehlsarchitektur: `syscall`

## 3.5 Der Betriebssystemkern (Kernel)

- „Eingang“  
Wo nimmt der Kernel den `syscall`-Befehl entgegen?
- „Ausgang“  
Wo ruft der Kernel die Callback-Funktionen der Treibermodule auf?
- „Weg durch den Kernel“  
Was passiert dazwischen?

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

19. Mai 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

## 1 Einführung

## 2 Unix

## 3 Treiberentwicklung

3.0 Massenspeicher und Gerätedateien

3.1 Mikrocontroller

3.2 Betriebssysteme ohne Speicherschutz

3.3 Linux-Kernel-Module

3.4 Die Systembibliothek (libc)

3.5 Der Betriebssystemkern (Kernel)

## 4 Speicherverwaltung

...

## 3.3 Linux-Kernel-Module

- **#include** <linux/module.h>  
**#include** <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE\_LICENSE()**
- Kommunikation mit Anwendungen:  
Geräte-Dateien (*major/minor number*)
- Kommunikation mit dem Betriebssystemkern:  
Callback-Funktionen (virtuelle Methoden)
- Automatisches Anlegen von Gerätedateien:  
Geräteklassen



## 3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf  
Indirekter Funktionsaufruf über eine Tabelle von Zeigern, abhängig von der Datei  
—→ Aufruf einer virtuellen Methode (objektorientiert)
- Verbindung zum Betriebssystemkern: System Call  
Das Betriebssystem hat mehr Rechte als die Anwendung, daher darf die Anwendung Funktionen des Betriebssystems nur auf sehr kontrollierte Weise aufrufen.  
—→ Hardware-Unterstützung  
Beispiel: x64-Befehlsarchitektur: `syscall`

## 3.5 Der Betriebssystemkern (Kernel)

- „Eingang“  
Wo nimmt der Kernel den `syscall`-Befehl entgegen?
- „Ausgang“  
Wo ruft der Kernel die Callback-Funktionen der Treibermodule auf?  
`ksys_write()`  
`vfs_write()`
- „Weg durch den Kernel“  
Was passiert dazwischen?

## 4 Speicherverwaltung

### 4.1 Microcontroller

Speicherverwaltung „einfach so“

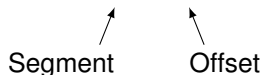
- PC erzeugt Speicher-Layout (z. B. Intel-[.hex](#)-Datei)
- Darin enthalten: Programm, Interrupt-Vektoren, ...
- Aufspielen in den Flash-Speicher

# 4 Speicherverwaltung

## 4.2 Speichersegmentierung

Intel 8086: Segment- und Offset-Adresse

- Beispiel: Speicherzelle **28F5:0100**: für das aktuelle Programm vorgesehen

  
Segment                      Offset

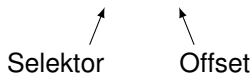
- Speicherzelle **B800:0000**: Bildschirmspeicher (Textmodus)
- physikalische Adresse =  $0x10 \cdot \text{Segment} + \text{Offset}$ , hier also: **B8000**
- Speicher insgesamt: 5 Hex-Ziffern = 20 Bit für Speicher-Adressierung  
→ maximal 1 MiB adressierbar
- „Speicherschutz“: Das Betriebssystem weist jedem Programm ein Segment zu (hier z. B.: **28F5**). Außerhalb davon **sollte** man nicht schreiben.  
(→ effiziente Bildschirmausgabe: direkt in Segment **B800** schreiben)

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80286: Selektoren

- Beispiel: Speicherzelle **0007:0100**: für das aktuelle Programm vorgesehen

  
Selektor                      Offset

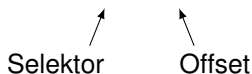
- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: **28F5**) des Segments
  - Länge des Segments (maximal 64 kiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- In der Praxis (MS-DOS): *kein* Speicherschutz, sondern:  
Jedes Programm verwaltet die Deskriptorentabelle selbst.

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80386: 32-Bit-Offsets

- Beispiel: Speicherzelle 0007:00000100



- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: 28F5) des Segments
  - Länge des Segments (maximal 4 GiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- MS-DOS und MS-Windows bis Version ME: *kein* Speicherschutz, sondern: Jedes Programm verwaltet die Deskriptorentabelle selbst.
- MS-Windows ab Version NT, OS/2, MacOS, Linux und andere Unixe: Speicherschutz mit Hardware-Unterstützung

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

#### Speicherschutz mit Hardware-Unterstützung

- spezielle Variable („Register“) im Prozessor:  
Berechtigungsstufe des aktuell laufenden Programms
- Benutzerprogramme dürfen Befehle zur Manipulation der Deskriptorentabelle nicht ausführen. Dies darf nur der Betriebssystemkern.
- Speicherzugriffe außerhalb des zugewiesenen Segments lösen eine *Exception* aus (ähnlich Interrupt). Das Betriebssystem kann daraufhin das Programm kontrolliert beenden („Speicherzugriffsfehler“).
- Das Betriebssystem kann Code als „nicht ausführbar“ markieren. Versuche, ihn auszuführen, lösen eine Exception aus.

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.  
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren



## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.  
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren
- Weiterhin möglich: *return-oriented programming*

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

26. Mai 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

**1 Einführung**

**2 Unix**

**3 Treiberentwicklung**

**4 Speicherverwaltung**

4.1 Mikrocontroller

4.2 Speichersegmentierung

4.3 Speicherschutz (*Protected Mode*)

4.4 Virtueller Speicher

**5 Dateisysteme**

...

## 4 Speicherverwaltung

### 4.1 Microcontroller

Speicherverwaltung „einfach so“

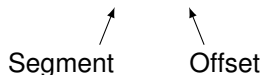
- PC erzeugt Speicher-Layout (z. B. Intel-[.hex](#)-Datei)
- Darin enthalten: Programm, Interrupt-Vektoren, ...
- Aufspielen in den Flash-Speicher

# 4 Speicherverwaltung

## 4.2 Speichersegmentierung

Intel 8086: Segment- und Offset-Adresse

- Beispiel: Speicherzelle **28F5:0100**: für das aktuelle Programm vorgesehen

  
Segment                  Offset

- Speicherzelle **B800:0000**: Bildschirmspeicher (Textmodus)
- physikalische Adresse =  $0x10 \cdot \text{Segment} + \text{Offset}$ , hier also: **B8000**
- Speicher insgesamt: 5 Hex-Ziffern = 20 Bit für Speicher-Adressierung  
→ maximal 1 MiB adressierbar
- „Speicherschutz“: Das Betriebssystem weist jedem Programm ein Segment zu (hier z. B.: **28F5**). Außerhalb davon **sollte** man nicht schreiben.  
(→ effiziente Bildschirmausgabe: direkt in Segment **B800** schreiben)

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80286: Selektoren

- Beispiel: Speicherzelle **0007:0100**: für das aktuelle Programm vorgesehen

                  ↑                  ↑  
          Selektor          Offset

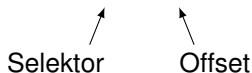
- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: **28F5**) des Segments
  - Länge des Segments (maximal 64 kiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- In der Praxis (MS-DOS): *kein* Speicherschutz, sondern:  
Jedes Programm verwaltet die Deskriptorentabelle selbst.

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80386: 32-Bit-Offsets

- Beispiel: Speicherzelle 0007:00000100



- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: 28F5) des Segments
  - Länge des Segments (maximal 4 GiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- MS-DOS und MS-Windows bis Version ME: *kein* Speicherschutz, sondern: Jedes Programm verwaltet die Deskriptorentabelle selbst.
- MS-Windows ab Version NT, OS/2, MacOS, Linux und andere Unixe: Speicherschutz mit Hardware-Unterstützung

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

#### Speicherschutz mit Hardware-Unterstützung

- spezielle Variable („Register“) im Prozessor:  
Berechtigungsstufe des aktuell laufenden Programms
- Benutzerprogramme dürfen Befehle zur Manipulation der Deskriptorentabelle nicht ausführen. Dies darf nur der Betriebssystemkern.
- Speicherzugriffe außerhalb des zugewiesenen Segments lösen eine *Exception* aus (ähnlich Interrupt). Das Betriebssystem kann daraufhin das Programm kontrolliert beenden („Speicherzugriffsfehler“).
- Das Betriebssystem kann Code als „nicht ausführbar“ markieren. Versuche, ihn auszuführen, lösen eine Exception aus.



## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.  
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.  
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren
- Weiterhin möglich: *return-oriented programming*

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Untersützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Unterstützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen
- Weitere Anwendungen:
  - *Shared Memory*
  - Datei in Speicher abbilden
  - kontrollierter Direktzugriff auf Speicher

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Untersützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen
- Weitere Anwendungen:
  - *Shared Memory*
  - Datei in Speicher abbilden
  - kontrollierter Direktzugriff auf Speicher
- Hardware-Fehler können zu Sicherheitslücken führen,  
z. B. *Meltdown* (2017)

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Untersützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen
- Weitere Anwendungen:
  - *Shared Memory*
  - Datei in Speicher abbilden
  - kontrollierter Direktzugriff auf Speicher
- Hardware-Fehler können zu Sicherheitslücken führen,  
z. B. *Meltdown* (2017)

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

2. Juni 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

**1 Einführung**

**2 Unix**

**3 Treiberentwicklung**

**4 Speicherverwaltung**

**4.1** Mikrocontroller

**4.2** Speichersegmentierung

**4.3** Speicherschutz (*Protected Mode*)

**4.4** Virtueller Speicher

**5 Dateisysteme**

...



## 4 Speicherverwaltung

### 4.1 Microcontroller

Speicherverwaltung „einfach so“

- PC erzeugt Speicher-Layout (z. B. Intel-[.hex](#)-Datei)
- Darin enthalten: Programm, Interrupt-Vektoren, ...
- Aufspielen in den Flash-Speicher

# 4 Speicherverwaltung

## 4.2 Speichersegmentierung

Intel 8086: Segment- und Offset-Adresse

- Beispiel: Speicherzelle **28F5:0100**: für das aktuelle Programm vorgesehen

                    ↑                    ↑  
                  Segment          Offset

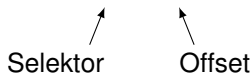
- Speicherzelle **B800:0000**: Bildschirmspeicher (Textmodus)
- physikalische Adresse =  $0x10 \cdot \text{Segment} + \text{Offset}$ , hier also: **B8000**
- Speicher insgesamt: 5 Hex-Ziffern = 20 Bit für Speicher-Adressierung  
→ maximal 1 MiB adressierbar
- „Speicherschutz“: Das Betriebssystem weist jedem Programm ein Segment zu (hier z. B.: **28F5**). Außerhalb davon **sollte** man nicht schreiben.  
(→ effiziente Bildschirmausgabe: direkt in Segment **B800** schreiben)

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80286: Selektoren

- Beispiel: Speicherzelle **0007:0100**: für das aktuelle Programm vorgesehen

  
Selektor                      Offset

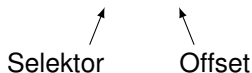
- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: **28F5**) des Segments
  - Länge des Segments (maximal 64 kiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- In der Praxis (MS-DOS): *kein* Speicherschutz, sondern:  
Jedes Programm verwaltet die Deskriptorentabelle selbst.

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Intel 80386: 32-Bit-Offsets

- Beispiel: Speicherzelle 0007:00000100



- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*  
Segment-Deskriptor:
  - physikalische Adresse (z. B.: 28F5) des Segments
  - Länge des Segments (maximal 4 GiB)
  - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- MS-DOS und MS-Windows bis Version ME: *kein* Speicherschutz, sondern: Jedes Programm verwaltet die Deskriptorentabelle selbst.
- MS-Windows ab Version NT, OS/2, MacOS, Linux und andere Unixe: Speicherschutz mit Hardware-Unterstützung

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

#### Speicherschutz mit Hardware-Unterstützung

- spezielle Variable („Register“) im Prozessor:  
Berechtigungsstufe des aktuell laufenden Programms
- Benutzerprogramme dürfen Befehle zur Manipulation der Deskriptorentabelle nicht ausführen. Dies darf nur der Betriebssystemkern.
- Speicherzugriffe außerhalb des zugewiesenen Segments lösen eine *Exception* aus (ähnlich Interrupt). Das Betriebssystem kann daraufhin das Programm kontrolliert beenden („Speicherzugriffsfehler“).
- Das Betriebssystem kann Speicherbereiche als „nicht ausführbar“ markieren. Versuche, dort Code auszuführen, lösen eine Exception aus.

## 4 Speicherverwaltung

### 4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.  
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren
- Weiterhin möglich: *return-oriented programming*

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Untersützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen
- Weitere Anwendungen:
  - *Shared Memory*
  - Datei in Speicher abbilden
  - kontrollierter Direktzugriff auf Speicher
- Hardware-Fehler können zu Sicherheitslücken führen,  
z. B. *Meltdown* (2017)

## 5 Dateisysteme

MS-DOS und kompatible Systeme: FAT-Dateisysteme  
„kleinster gemeinsamer Nenner“

Unterteilung des Datenträgers in *Cluster*

*File Allocation Table (FAT)*

- Belegung der Cluster
- Dateien: verkettete Listen

Verzeichnisse

- Dateiname
- Meta-Daten
- Start-Cluster

Besonderheiten

- starke Fragmentierung
- Dateinamen: 8.3 + zusätzliche Einträge
- begrenztes Wurzelverzeichnis
- Zeit-Auflösung: 2 Sekunden



# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

16. Juni 2025

# Treiberentwicklung, Echtzeit- und Betriebssysteme

## 1 Einführung

## 2 Unix

## 3 Treiberentwicklung

## 4 Speicherverwaltung

### 4.1 Mikrocontroller

### 4.2 Speichersegmentierung

### 4.3 Speicherschutz (*Protected Mode*)

### 4.4 Virtueller Speicher

## 5 Dateisysteme

### 5.1 FAT

### 5.2 Unix-Dateisysteme

...

# 4 Speicherverwaltung

## 4.4 Virtueller Speicher

Hardware-Untersützung durch *Memory Management Unit (MMU)*

- Unterteilung des Speichers in *Seiten* (typischerweise: 4 kiB)
- Zeiger: Zugriff auf *virtuellen Speicher*
- Tabelle innerhalb der MMU: Zuordnung der Seiten zu physikalischem oder nicht zugeordnetem Speicher
- Anwendung: ausgelagerter Speicher, bei Zugriff: Exception  
Betriebssystem kann ausgelagerte Seite bereitstellen
- Weitere Anwendungen:
  - *Shared Memory*
  - Datei in Speicher abbilden
  - kontrollierter Direktzugriff auf Speicher
- Hardware-Fehler können zu Sicherheitslücken führen,  
z. B. *Meltdown* (2017)

# 5 Dateisysteme

## 5.1 FAT

MS-DOS und kompatible Systeme, „kleinster gemeinsamer Nenner“

Unterteilung des Datenträgers in *Cluster*

*File Allocation Table (FAT)*

- Belegung der Cluster
- Dateien: verkettete Listen

Verzeichnisse

- Dateiname
- Meta-Daten
- Start-Cluster

Besonderheiten

- starke Fragmentierung
- Dateinamen: 8.3 + zusätzliche Einträge
- begrenztes Wurzelverzeichnis
- Zeit-Auflösung: 2 Sekunden

## 5.2 Unix-Dateisysteme

Unix-kompatible Dateisysteme: *Index Nodes (inodes)*

Unterteilung des Datenträgers in *Blöcke*

*Superblock*

- Meta-Daten zum Datenträger
- Zeiger auf Wurzelverzeichnis

Verzeichnisse

- Dateiname
- Zeiger auf inode

inode

- Meta-Daten zur Datei
- direkte und indirekte Zeiger auf Blöcke

Besonderheiten

- Fragmentierung i. d. R. kein Problem
- Der inode „ist“ die Datei.
- Pseudo-Dateien oder sehr kurze Dateien:  
Daten direkt im inode