

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

3. November 2020

# Was sind eingebettete Systeme?

*Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.*

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller



# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)
- Programmierung oft außerhalb des Systems (Cross-Entwicklungswerkzeuge)

# In dieser Lehrveranstaltung

- Einführung in Unix
- TCP/IP und Bus-Systeme in der Praxis
- C-Programmierung für Fortgeschrittene
- Echtzeit-Systeme in Theorie und Praxis
- Prüfungsleistung: Projektaufgabe  
Eingebettetes System eigener Wahl zum Laufen bringen

→ **Projektaufgabe überlegen**

# Eingebettete Systeme

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

## 1 Einführung

1.1 Was sind eingebettete Systeme?

1.2 In dieser Lehrveranstaltung

## 2 Einführung in 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 TCP/IP in der Praxis

...

## 2 Einführung in Unix

### 2.1 Grundkonzepte

- 1965** Vorgänger: Multics (Multiplexed Information and Computing Service)  
„überladen“
- 1970** Unix: Einfachheit als Grundkonzept
- 1972** Umstellung auf neu entwickelte Programmiersprache C
- 1975** AT&T: Unix inkl. Quelltext für Universitäten
- 1977** Berkeley Software Distribution (BSD)
- 1983** GNU-Projekt
- 1987** Minix
- 1991** Linux
- 1993** FreeBSD, NetBSD
- 1994** OpenBSD
- 2000** Darwin (Mac OS X, BSD-basiert)
- 2008** Android (Linux-basiert)

## 2 Einführung in Unix

### 2.1 Grundkonzepte

Unix und C: Einfachheit als Grundkonzept

- Vermeiden von Ausnahmen
- Baukastensystem

C: Hauptprogramm  
= „normale“ Funktion

```
int main (int argc, char **argv)
{
    printf ("Hello, _world!\n");
    return 0;
}
```

Unix: übergeordnetes Verzeichnis = „normales“ Verzeichnis

```
cassini/home/peter/foo> ls -la
insgesamt 24
drwxr-xr-x  2 peter peter  4096 Okt  6 13:30 .
drwxr-xr-x 172 peter peter 20480 Okt  6 13:30 ..
cassini/home/peter/foo> cd ..
cassini/home/peter>
```

## 2 Einführung in Unix

### 2.1 Grundkonzepte

Unix und C: Einfachheit als Grundkonzept

- Vermeiden von Ausnahmen
- Baukastensystem

C: Bibliotheken

z. B.: `printf()` = „normale“ Funktion  
aus eine Bibliothek (`libc`)

Unix: Programme arbeiten zusammen

```
cassini/home/peter/bo> find . -name "*klausur*.tex" \  
| xargs grep -l "PBM-Datei"  
./2014ws/ainf/20150130.0/ainf-klausur-20150130.tex  
./2016ws/hp/20170920.0/klausur.tex  
./2016ws/hp/20170206.0/klausur.tex  
./2011ws/rarch/20120322.0/rarch-klausur-20120322.tex  
./2012ws/klausuren-gerwinski/rarch-klausur-20120322.tex  
./2013ws/ainf/20140918.0/ainf-klausur-20140918.tex  
./2017ws/hp/20180213.k1/klausur.tex  
./2017ws/hp/20180205/klausur.tex  
./2015ws/ainf/20160913/ainf-klausur-20160913.tex
```



## 2.2 Die Kommandozeile: Grundlagen

- Programm aufrufen: Namen eingeben, z. B.: `ls`
- Optionen: `ls -l`
- Lange Optionen (GNU-Konvention): `ls --help`
- Text schreiben: `echo "Hello, world!"`
- (String-)Variable setzen: `FOO=bar`
- Variable abrufen: `echo $FOO`

## 2.2 Die Kommandozeile: Grundlagen

- Programm aufrufen: Namen eingeben, z. B.: `ls`
- Optionen: `ls -l`
- Lange Optionen (GNU-Konvention): `ls --help`
- Text schreiben: `echo "Hello, world!"`
- (String-)Variable setzen: `FOO=bar`
- Variable abrufen: `echo $FOO`

```
cassini/home/peter/bo> FOO=ls
cassini/home/peter/bo> echo $FOO
ls
cassini/home/peter/bo> $FOO
2011ws  2012ws  2013ws  doc          misc  projekte
2012ss  2013ss  briefe  material    orga
cassini/home/peter/bo>
```

## 2.2 Die Kommandozeile: Grundlagen

- Befehl zurückholen: Pfeiltasten  $\uparrow$ ,  $\downarrow$
- Befehl bearbeiten: Pfeiltasten  $\leftarrow$ ,  $\rightarrow$  usw.
- Befehl vervollständigen: TAB
- Befehl rückwärts suchen: Ctrl+R
- Bildschirm löschen: Ctrl+L
- Befehl abbrechen: Ctrl+C

## 2.2 Die Kommandozeile: Grundlagen

- Befehl zurückholen: Pfeiltasten  $\uparrow$ ,  $\downarrow$
- Befehl bearbeiten: Pfeiltasten  $\leftarrow$ ,  $\rightarrow$  usw.
- Befehl vervollständigen: TAB
- Befehl rückwärts suchen: Ctrl+R
- Bildschirm löschen: Ctrl+L
- Befehl abbrechen: Ctrl+C
  
- Hilfe-Option: `ls --help`
- Unix-Handbuch – *manual*: `man ls`  
(Beenden mit `q`)

## 2.2 Die Kommandozeile: Grundlagen

- Verzeichnisse für Programme: `echo $PATH`
- Programm in explizitem Verzeichnis aufrufen: `/bin/ls -l`
- Programm im aktuellen Verzeichnis aufrufen: `./hello`

**MS-DOS:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen  
*oder* sich im aktuellen Verzeichnis befinden.

**Unix:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen.  
→ Vermeiden von Ausnahmen

## 2.2 Die Kommandozeile: Grundlagen

- Verzeichnisse für Programme: `echo $PATH`
- Programm in explizitem Verzeichnis aufrufen: `/bin/ls -l`
- Programm im aktuellen Verzeichnis aufrufen: `./hello`

**MS-DOS:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen  
*oder* sich im aktuellen Verzeichnis befinden.

**Unix:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen.

—→ Vermeiden von Ausnahmen

Das aktuelle Verzeichnis ( `.` ) *kann* im `PATH` stehen,  
muß dies aber nicht –  
und sollte es aus Sicherheitsgründen auch nicht.

## 2.3 Dateisysteme

- Dateien listen: `ls`  
langes Listenformat: `ls -l`  
rückwärts nach Zeit sortiert: `ls -lrt`
- Datei ausgeben: `cat hello.c`
- Datei anzeigen: `less hello.c`

## 2.3 Dateisysteme

- Arbeitsverzeichnis anzeigen: `pwd`
- Arbeitsverzeichnis wechseln: `cd script`  
(*kein* Programm, sondern Shell-Befehl)
- übergeordnetes Verzeichnis: `cd ..`
- eigenes *Home*-Verzeichnis: `cd`
- Wurzelverzeichnis: `cd /`
- wieder zurück: `cd -`



## 2.3 Dateisysteme

- Arbeitsverzeichnis anzeigen: `pwd`
- Arbeitsverzeichnis wechseln: `cd script`  
(*kein* Programm, sondern Shell-Befehl)
- übergeordnetes Verzeichnis: `cd ..`
- eigenes *Home*-Verzeichnis: `cd`
- Wurzelverzeichnis: `cd /`
- wieder zurück: `cd -`

```
cassini/home/peter/bo/2013ss/net/script> cd /usr/bin
cassini/usr/bin> cd ../lib
cassini/usr/lib> cd
cassini/home/peter>
```

## 2.3 Dateisysteme

- Dateien kopieren (*copy*): `cp`
- Dateien verschieben/umbenennen (*move*): `mv`
- Dateien löschen (*remove*): `rm`

```
cassini/home/peter> cp -p foo/test.txt
cp: missing destination file operand after 'foo/test.txt'
Try 'cp --help' for more information.
cassini/home/peter> cp -p foo/test.txt .
cassini/home/peter> mv test.txt bla.txt
cassini/home/peter> cat bla.txt
Dies ist ein Test.
cassini/home/peter> rm bla.txt
cassini/home/peter>
```

Aktuelles Verzeichnis: `.`

## 2.3 Dateisysteme

- Datenträger in Verzeichnis *einhängen*: `mount`

```
cassini/home/peter> ls /media/usb1/  
cassini/home/peter> mount /media/usb1  
cassini/home/peter> ls /media/usb1/  
es-20181018.pdf  hello.c  hexapode  KIS-Bericht.pdf  
cassini/home/peter> umount /media/usb1  
cassini/home/peter> ls /media/usb1/  
cassini/home/peter>
```

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter    1539 Nov 29  2012 orbit-x1.c
```

## 2.3 Dateisysteme

- Zugriffsrechte

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter      1539 Nov 29  2012 orbit-x1.c
```




Benutzer (u – user) darf lesen und schreiben

## 2.3 Dateisysteme

- Zugriffsrechte

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter      1539 Nov 29  2012 orbit-x1.c
```



Gruppe (g – group) darf lesen

## 2.3 Dateisysteme

- Zugriffsrechte

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter      1539 Nov 29  2012 orbit-x1.c
```



alle anderen (o – *other*) dürfen lesen

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter      1539 Nov 29  2012 orbit-xl.c
```

- Zugriffsrechte ändern:

`chmod o-r orbit-xl.c` – Lesezugriff entziehen

`chmod g+w orbit-xl.c` – Schreibzugriff gewähren

`chmod 640 orbit-xl.c` – auf `-rw-r-----` setzen



## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2013ws/ainf/20131031.0> ls -l
...
-rw-r--r-- 1 peter peter      1539 Nov 29  2012 orbit-xl.c
```

- Zugriffsrechte ändern:

`chmod o-r orbit-xl.c` – Lesezugriff entziehen

`chmod g+w orbit-xl.c` – Schreibzugriff gewähren

`chmod 640 orbit-xl.c` – auf `-rw-r-----` setzen

6 4 0

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2013ws/systech/20131008.0> cat test
ls -l systech-20131008.*
cassini/home/peter/bo/2013ws/systech/20131008.0> chmod +x test
cassini/home/peter/bo/2013ws/systech/20131008.0> ls -l test
-rwxr-xr-x 1 peter peter 25 Okt 6 16:45 test
cassini/home/peter/bo/2013ws/systech/20131008.0> ./test
-rw-r--r-- 1 peter peter 4120 Okt 6 16:44 systech-20131008.aux
...
```

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2013ws/systech/20131008.0> cat test
ls -l systech-20131008.*
cassini/home/peter/bo/2013ws/systech/20131008.0> chmod +x test
cassini/home/peter/bo/2013ws/systech/20131008.0> ls -l test
-rwxr-xr-x 1 peter peter 25 Okt 6 16:45 test
cassini/home/peter/bo/2013ws/systech/20131008.0> ./test
-rw-r--r-- 1 peter peter 4120 Okt 6 16:44 systech-20131008.aux
...
```

- ausführbare Textdateien: *Skripte*

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2013ws/systech/20131008.0> cat test
ls -l systech-20131008.*
cassini/home/peter/bo/2013ws/systech/20131008.0> chmod +x test
cassini/home/peter/bo/2013ws/systech/20131008.0> ls -l test
-rwxr-xr-x 1 peter peter 25 Okt 6 16:45 test
cassini/home/peter/bo/2013ws/systech/20131008.0> ./test
-rw-r--r-- 1 peter peter 4120 Okt 6 16:44 systech-20131008.aux
...
```

- ausführbare Textdateien: *Skripte*

hier: ausführbare Textdatei mit Shell-Befehlen  
(ohne spezielle Kennung): Shell-Skript

# Eingebettete Systeme

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

## 1 Einführung

1.1 Was sind eingebettete Systeme?

1.2 In dieser Lehrveranstaltung

## 2 Einführung in 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 TCP/IP in der Praxis

...

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

10. November 2020

# Eingebettete Systeme

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

## 1 Einführung

- 1.1 Was sind eingebettete Systeme?
- 1.2 Vertiefung Systemtechnik
- 1.3 In dieser Lehrveranstaltung

## 2 Einführung in 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 TCP/IP in der Praxis

...

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. Flugsimulator, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)
- Programmierung oft außerhalb des Systems (Cross-Entwicklungswerkzeuge)



# In dieser Lehrveranstaltung

- Einführung in Unix
- TCP/IP und Bus-Systeme in der Praxis
- C-Programmierung für Fortgeschrittene
- Echtzeit-Systeme in Theorie und Praxis
- Prüfungsleistung: Projektaufgabe  
Eingebettetes System eigener Wahl zum Laufen bringen

→ **Projektaufgabe überlegen**

Ideen:

- Steuerung von Robotern
- Smart Home
- Smartphone im Selbstbau
- Verbesserungen an unseren Online-Werkzeugen  
für Lehre und Home-Office

## 2 Einführung in Unix

### 2.1 Grundkonzepte

- 1965** Vorgänger: Multics (Multiplexed Information and Computing Service)  
„überladen“
- 1970** Unix: Einfachheit als Grundkonzept
- 1972** Umstellung auf neu entwickelte Programmiersprache C
- 1975** AT&T: Unix inkl. Quelltext für Universitäten
- 1977** Berkeley Software Distribution (BSD)
- 1983** GNU-Projekt
- 1987** Minix
- 1991** Linux
- 1993** FreeBSD, NetBSD
- 1994** OpenBSD
- 2000** Darwin (Mac OS X, BSD-basiert)
- 2008** Android (Linux-basiert)

## 2 Einführung in Unix

### 2.1 Grundkonzepte

Unix und C: Einfachheit als Grundkonzept

- Vermeiden von Ausnahmen
- Baukastensystem

C: Hauptprogramm  
= „normale“ Funktion

```
int main (int argc, char **argv)
{
    printf ("Hello, _world!\n");
    return 0;
}
```

Unix: übergeordnetes Verzeichnis = „normales“ Verzeichnis

```
cassini/home/peter/foo> ls -la
insgesamt 24
drwxr-xr-x  2 peter peter  4096 Okt  6 13:30 .
drwxr-xr-x 172 peter peter 20480 Okt  6 13:30 ..
cassini/home/peter/foo> cd ..
cassini/home/peter>
```

## 2 Einführung in Unix

### 2.1 Grundkonzepte

Unix und C: Einfachheit als Grundkonzept

- Vermeiden von Ausnahmen
- Baukastensystem

C: Bibliotheken

z. B.: `printf()` = „normale“ Funktion  
aus eine Bibliothek (`libc`)

Unix: Programme arbeiten zusammen

```
cassini/home/peter/bo> find . -name "*klausur*.tex" \  
| xargs grep -l "PBM-Datei"  
./2014ws/ainf/20150130.0/ainf-klausur-20150130.tex  
./2016ws/hp/20170920.0/klausur.tex  
./2016ws/hp/20170206.0/klausur.tex  
./2011ws/rarch/20120322.0/rarch-klausur-20120322.tex  
./2012ws/klausuren-gerwinski/rarch-klausur-20120322.tex  
./2013ws/ainf/20140918.0/ainf-klausur-20140918.tex  
./2017ws/hp/20180213.k1/klausur.tex  
./2017ws/hp/20180205/klausur.tex  
./2015ws/ainf/20160913/ainf-klausur-20160913.tex
```

## 2.2 Die Kommandozeile: Grundlagen

- Programm aufrufen: Namen eingeben, z. B.: `ls`
- Optionen: `ls -l`
- Lange Optionen (GNU-Konvention): `ls --help`
- Text schreiben: `echo "Hello, world!"`
- (String-)Variable setzen: `FOO=bar`
- Variable abrufen: `echo $FOO`

```
cassini/home/peter/bo> FOO=ls
cassini/home/peter/bo> echo $FOO
ls
cassini/home/peter/bo> $FOO
2011ws  2012ws  2013ws  doc          misc  projekte
2012ss  2013ss  briefe  material    orga
cassini/home/peter/bo>
```

## 2.2 Die Kommandozeile: Grundlagen

- Befehl zurückholen: Pfeiltasten  $\uparrow$ ,  $\downarrow$
- Befehl bearbeiten: Pfeiltasten  $\leftarrow$ ,  $\rightarrow$  usw.
- Befehl vervollständigen: TAB
- Befehl rückwärts suchen: Ctrl+R
- Bildschirm löschen: Ctrl+L
- Befehl abbrechen: Ctrl+C
  
- Hilfe-Option: `ls --help`
- Unix-Handbuch – *manual*: `man ls`  
(Beenden mit `q`)

## 2.2 Die Kommandozeile: Grundlagen

- Verzeichnisse für Programme: `echo $PATH`
- Programm in explizitem Verzeichnis aufrufen: `/bin/ls -l`
- Programm im aktuellen Verzeichnis aufrufen: `./hello`

**MS-DOS:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen  
oder sich im aktuellen Verzeichnis befinden.

**Unix:** Ausführbare Programme werden gefunden,  
wenn sie im `PATH` stehen.

—→ Vermeiden von Ausnahmen

Das aktuelle Verzeichnis ( `.` ) *kann* im `PATH` stehen,  
muß dies aber nicht –  
und sollte es aus Sicherheitsgründen auch nicht.

## 2.3 Dateisysteme

- Dateien listen: `ls`  
langes Listenformat: `ls -l`  
rückwärts nach Zeit sortiert: `ls -lrt`
- Datei ausgeben: `cat hello.c`
- Datei anzeigen: `less hello.c`



## 2.3 Dateisysteme

- Arbeitsverzeichnis anzeigen: `pwd`
- Arbeitsverzeichnis wechseln: `cd script`  
(*kein* Programm, sondern Shell-Befehl)
- übergeordnetes Verzeichnis: `cd ..`
- eigenes *Home*-Verzeichnis: `cd`
- Wurzelverzeichnis: `cd /`
- wieder zurück: `cd -`

```
cassini/home/peter/bo/2013ss/net/script> cd /usr/bin
cassini/usr/bin> cd ../lib
cassini/usr/lib> cd
cassini/home/peter>
```

## 2.3 Dateisysteme

- Dateien kopieren (*copy*): `cp`
- Dateien verschieben/umbenennen (*move*): `mv`
- Dateien löschen (*remove*): `rm`

```
cassini/home/peter> cp -p foo/test.txt
cp: missing destination file operand after 'foo/test.txt'
Try 'cp --help' for more information.
cassini/home/peter> cp -p foo/test.txt .
cassini/home/peter> mv test.txt bla.txt
cassini/home/peter> cat bla.txt
Dies ist ein Test.
cassini/home/peter> rm bla.txt
cassini/home/peter>
```

Aktuelles Verzeichnis: `.`

## 2.3 Dateisysteme

- Datenträger in Verzeichnis *einhängen*: `mount`

```
cassini/home/peter> ls /media/usb1/  
cassini/home/peter> mount /media/usb1  
cassini/home/peter> ls /media/usb1/  
es-20191002.pdf  hello.c  hexapode  KIS-Bericht.pdf  
cassini/home/peter> umount /media/usb1  
cassini/home/peter> ls /media/usb1/  
cassini/home/peter>
```

## 2.3 Dateisysteme


- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```




Benutzer (u – *user*) darf lesen und schreiben

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```



Gruppe (g – *group*) darf lesen

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```



alle anderen (o – *other*) dürfen lesen

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

- Zugriffsrechte ändern:

```
chmod o-r es-20191009.tex – Lesezugriff entziehen
chmod g+w es-20191009.tex – Schreibzugriff gewähren
chmod 640 es-20191009.tex – auf -rw-r----- setzen
```



## 2.3 Dateisysteme

- Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

- Zugriffsrechte ändern:

```
chmod o-r es-20191009.tex – Lesezugriff entziehen
chmod g+w es-20191009.tex – Schreibzugriff gewähren
chmod 640 es-20191009.tex – auf -rw-r-- setzen
                        6   4   0
```

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2019ws/es/20191002> cat test2.txt
ls -l
cassini/home/peter/bo/2019ws/es/20191002> chmod +x test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ls -l test2.txt
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ./test2.txt
insgesamt 4828
lrwxrwxrwx 1 peter peter      18 Apr 13  2016 csa2.jpg -> ..
-rw-r--r-- 1 peter peter 4619138 Okt  8 21:28 es-20191002.pdf
...
```

- ausführbare Textdateien: *Skripte*

hier: ausführbare Textdatei mit Shell-Befehlen  
(ohne spezielle Kennung): Shell-Skript

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2019ws/es/20191002> cat test2.txt
ls -l
cassini/home/peter/bo/2019ws/es/20191002> chmod +x test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ls -l test2.txt
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ./test2.txt
insgesamt 4828
lrwxrwxrwx 1 peter peter      18 Apr 13  2016 csa2.jpg -> ..
-rw-r--r-- 1 peter peter 4619138 Okt  8 21:28 es-20191002.pdf
...
```

- ausführbare Textdateien: *Skripte*

hier: ausführbare Textdatei mit Shell-Befehlen  
(ohne spezielle Kennung): Shell-Skript

Kennung: 1. Zeile enthält `#!` und den Interpreter,  
z. B. `#!/bin/bash`

# Eingebettete Systeme

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

## 1 Einführung

- 1.1 Was sind eingebettete Systeme?
- 1.2 Vertiefung Systemtechnik
- 1.3 In dieser Lehrveranstaltung

## 2 Einführung in 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 TCP/IP in der Praxis

...

## 2.3 Dateisysteme

- *Symbolische Verknüpfungen – symbolic links*

Verweis auf die eigentliche Datei

→ Wenn man die Datei löscht, zeigt der Link ins Leere.

Verknüpfung anlegen: `ln -s datei link`

(Richtung: wie bei `cp`)

Beispiel: `ln -s ../common/GNU-GPL-3 gpl.txt`

## 2.3 Dateisysteme

- *Symbolische Verknüpfungen – symbolic links*

Verweis auf die eigentliche Datei

→ Wenn man die Datei löscht, zeigt der Link ins Leere.

Verknüpfung anlegen: `ln -s datei link`  
(Richtung: wie bei `cp`)

Beispiel: `ln -s ../common/GNU-GPL-3 gpl.txt`

- *Harte Verknüpfungen – hard links*

Dieselben Daten auf dem Datenträger  
sind unter mehreren Namen verfügbar.

→ Wenn man einen löscht, sind die Daten noch da.

```
cassini/home/peter/bo/2019ws/es/20191002> ls -l
```

```
...
```

```
-rw-r--r-- 1 peter peter 1202 Okt 2 13:35 shell-06.tx
```

```
drwxr-xr-x 2 peter peter 4096 Okt 2 13:16 test
```



Anzahl der („harten“) Links auf diese Datei / dieses Verzeichnis

## 2.3 Dateisysteme

- `grep`: Dateien durchsuchen

```
cassini/home/peter/bo/2019ws/es/20191002> grep gcc *.txt  
shell-03.txt: cassini/...> gcc -Wall -O hello.c -o hello
```

## 2.3 Dateisysteme

- **find**: Dateien anhand ihrer Eigenschaften suchen

```
$ find . -name "*.txt"
```

```
./shell-06.txt
```

```
./shell-03.txt
```

```
./shell-05.txt
```

```
./test.txt
```

```
./test/test.txt
```

```
...
```

```
$ find . -name "*.txt" -perm /u+x
```

```
./test2.txt
```

```
$ find . -name "*.txt" -perm /u+x -exec ls -l {} \;
```

```
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 ./test2.txt
```



## 2.4 Ein- und Ausgabeströme

- Standard-Ausgabe in Datei umleiten

```
$ echo "Dies ist ein Test." > test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

## 2.4 Ein- und Ausgabeströme

- Standard-Ausgabe in Datei umleiten

```
$ echo "Dies ist ein Test." > test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

- Standard-Ausgabe an Datei anhängen

```
$ echo "Dies ist noch ein Test." >> test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

```
Dies ist noch ein Test.
```

## 2.4 Ein- und Ausgabeströme

- Fehler-Ausgabe in Datei umleiten

```
$ cat gibtsnicht.txt > fehler.txt
cat: gibtsnicht.txt: No such file or directory
$ cat fehler.txt
$ cat gibtsnicht.txt 2> fehler.txt
$ cat fehler.txt
cat: gibtsnicht.txt: No such file or directory
```

## 2.4 Ein- und Ausgabeströme

- Standard-Eingabe aus Datei lesen

```
$ bc
bc 1.06.95
Copyright [...] 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
2 + 2
4
$ echo "2 + 2" > test.bc
$ bc < test.bc
4
```

## 2.5 Pipes

Standard-Ausgabe von Programm A  
wird zu Standard-Eingabe von Programm B

```
$ echo "2 + 2" | bc  
4
```

→ sehr mächtiger „Baukasten“

## 2.5 Pipes

- `sed`: *stream editor*

Suchen und Ersetzen (und noch viel mehr)

```
$ echo "Schlimmer geht nimmer." | sed -e 's/nim/im/g'  
Schlimmer geht immer.
```

## 2.5 Pipes

- **grep**: Standard-Eingabe durchsuchen

```
$ ls | grep slides
```

```
pgslides.sty
```

```
$ ls *.pdf | grep -v logo
```

```
es-20191002.pdf
```

```
Zeichen_123.pdf
```

```
$ ls -l $(ls *.pdf | grep -v logo)
```

```
-rw-r--r-- 1 ... 4619138 Okt 8 21:28 es-20191002.pdf
```

```
lrwxrwxrwx 1 ...          25 Okt 3  2016 Zeichen_123.pdf -> ...
```

## 2.6 Verzweigungen und Schleifen

```
$ if grep Blubb test.txt; then echo "gefunden"; \  
  else echo "nicht gefunden"; fi  
nicht gefunden  
$ for x in foo bar baz; do echo $x; done  
foo  
bar  
baz
```



# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in 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 TCP/IP in der Praxis

...

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

17. November 2020

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in 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

2.7 Skript-Programmierung

## 3 TCP/IP in der Praxis

...

## 2.3 Dateisysteme

- Datenträger in Verzeichnis *einhängen*: `mount`

```
cassini/home/peter> ls /media/usb1/
cassini/home/peter> mount /media/usb1
cassini/home/peter> ls /media/usb1/
es-20191002.pdf  hello.c  hexapode  KIS-Bericht.pdf
cassini/home/peter> umount /media/usb1
cassini/home/peter> ls /media/usb1/
cassini/home/peter>
```

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```



Benutzer (u – *user*) darf lesen und schreiben

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```



Gruppe (g – *group*) darf lesen

## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```



alle anderen (o – *other*) dürfen lesen



## 2.3 Dateisysteme

- *Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

- Zugriffsrechte ändern:

```
chmod o-r es-20191009.tex – Lesezugriff entziehen
chmod g+w es-20191009.tex – Schreibzugriff gewähren
chmod 640 es-20191009.tex – auf -rw-r----- setzen
```

## 2.3 Dateisysteme

- Zugriffsrechte*

```
cassini/home/peter/bo/2019ws/es/20191009> ls -l
...
-rw-r--r-- 1 peter peter 24523 Okt  8 21:47 es-20191009.tex
```

- Zugriffsrechte ändern:

```
chmod o-r es-20191009.tex – Lesezugriff entziehen
chmod g+w es-20191009.tex – Schreibzugriff gewähren
chmod 640 es-20191009.tex – auf -rw-r-- setzen
                        6   4   0
```

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2019ws/es/20191002> cat test2.txt
ls -l
cassini/home/peter/bo/2019ws/es/20191002> chmod +x test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ls -l test2.txt
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ./test2.txt
insgesamt 4828
lrwxrwxrwx 1 peter peter      18 Apr 13  2016 csa2.jpg -> ..
-rw-r--r-- 1 peter peter 4619138 Okt  8 21:28 es-20191002.pdf
...
```

- ausführbare Textdateien: *Skripte*

hier: ausführbare Textdatei mit Shell-Befehlen  
(ohne spezielle Kennung): Shell-Skript

## 2.3 Dateisysteme

- *ausführbare* Dateien

```
cassini/home/peter/bo/2019ws/es/20191002> cat test2.txt
ls -l
cassini/home/peter/bo/2019ws/es/20191002> chmod +x test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ls -l test2.txt
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 test2.txt
cassini/home/peter/bo/2019ws/es/20191002> ./test2.txt
insgesamt 4828
lrwxrwxrwx 1 peter peter      18 Apr 13  2016 csa2.jpg -> ..
-rw-r--r-- 1 peter peter 4619138 Okt  8 21:28 es-20191002.pdf
...
```

- ausführbare Textdateien: *Skripte*

hier: ausführbare Textdatei mit Shell-Befehlen  
(ohne spezielle Kennung): Shell-Skript

Kennung: 1. Zeile enthält `#!` und den Interpreter,  
z. B. `#!/bin/bash`

# Eingebettete Systeme

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

## 1 Einführung

- 1.1 Was sind eingebettete Systeme?
- 1.2 Vertiefung Systemtechnik
- 1.3 In dieser Lehrveranstaltung

## 2 Einführung in 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 TCP/IP in der Praxis

...

## 2.3 Dateisysteme

- *Symbolische Verknüpfungen – symbolic links*

Verweis auf die eigentliche Datei

→ Wenn man die Datei löscht, zeigt der Link ins Leere.

Verknüpfung anlegen: `ln -s datei link`

(Richtung: wie bei `cp`)

Beispiel: `ln -s ../common/GNU-GPL-3 gpl.txt`

## 2.3 Dateisysteme

- *Symbolische Verknüpfungen – symbolic links*

Verweis auf die eigentliche Datei

→ Wenn man die Datei löscht, zeigt der Link ins Leere.

Verknüpfung anlegen: `ln -s datei link`  
(Richtung: wie bei `cp`)

Beispiel: `ln -s ../common/GNU-GPL-3 gpl.txt`

- *Harte Verknüpfungen – hard links*

Dieselben Daten auf dem Datenträger  
sind unter mehreren Namen verfügbar.

→ Wenn man einen löscht, sind die Daten noch da.

```
cassini/home/peter/bo/2019ws/es/20191002> ls -l
```

```
...
```

```
-rw-r--r-- 1 peter peter    1202 Okt  2 13:35 shell-06.tx  
drwxr-xr-x 2 peter peter   4096 Okt  2 13:16 test
```



Anzahl der („harten“) Links auf diese Datei / dieses Verzeichnis

## 2.3 Dateisysteme

- `grep`: Dateien durchsuchen

```
cassini/home/peter/bo/2019ws/es/20191002> grep gcc *.txt  
shell-03.txt: cassini/...> gcc -Wall -O hello.c -o hello
```



## 2.3 Dateisysteme

- **find**: Dateien anhand ihrer Eigenschaften suchen

```
$ find . -name "*.txt"
```

```
./shell-06.txt
```

```
./shell-03.txt
```

```
./shell-05.txt
```

```
./test.txt
```

```
./test/test.txt
```

```
...
```

```
$ find . -name "*.txt" -perm /u+x
```

```
./test2.txt
```

```
$ find . -name "*.txt" -perm /u+x -exec ls -l {} \;
```

```
-rwxr-xr-x 1 peter peter 6 Okt  2 13:43 ./test2.txt
```

## 2.4 Ein- und Ausgabeströme

- Standard-Ausgabe in Datei umleiten

```
$ echo "Dies ist ein Test." > test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

## 2.4 Ein- und Ausgabeströme

- Standard-Ausgabe in Datei umleiten

```
$ echo "Dies ist ein Test." > test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

- Standard-Ausgabe an Datei anhängen

```
$ echo "Dies ist noch ein Test." >> test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

```
Dies ist noch ein Test.
```

## 2.4 Ein- und Ausgabeströme

- Fehler-Ausgabe in Datei umleiten

```
$ cat gibtsnicht.txt > fehler.txt
cat: gibtsnicht.txt: No such file or directory
$ cat fehler.txt
$ cat gibtsnicht.txt 2> fehler.txt
$ cat fehler.txt
cat: gibtsnicht.txt: No such file or directory
```

## 2.4 Ein- und Ausgabeströme

- Standard-Eingabe aus Datei lesen

```
$ bc
bc 1.06.95
Copyright [...] 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
2 + 2
4
$ echo "2 + 2" > test.bc
$ bc < test.bc
4
```

## 2.5 Pipes

Standard-Ausgabe von Programm A  
wird zu Standard-Eingabe von Programm B

```
$ echo "2 + 2" | bc  
4
```

→ sehr mächtiger „Baukasten“

## 2.5 Pipes

- `sed`: *stream editor*

Suchen und Ersetzen (und noch viel mehr)

```
$ echo "Schlimmer geht nimmer." | sed -e 's/nim/im/g'  
Schlimmer geht immer.
```

## 2.5 Pipes

- **grep**: Standard-Eingabe durchsuchen

```
$ ls | grep slides
```

```
pgslides.sty
```

```
$ ls *.pdf | grep -v logo
```

```
es-20191002.pdf
```

```
Zeichen_123.pdf
```

```
$ ls -l $(ls *.pdf | grep -v logo)
```

```
-rw-r--r-- 1 ... 4619138 Okt 8 21:28 es-20191002.pdf
```

```
lrwxrwxrwx 1 ...          25 Okt 3  2016 Zeichen_123.pdf -> ...
```



## 2.6 Verzweigungen und Schleifen

```
$ if grep Blubb test.txt; then echo "gefunden"; \  
  else echo "nicht gefunden"; fi  
nicht gefunden  
$ for x in foo bar baz; do echo $x; done  
foo  
bar  
baz
```

## 2.7 Skript-Programmierung

**Aufgabe:** Programmieren Sie ein Unix-Shell-Skript, das etwas Nützliches macht. :-)

Beispiel: CSV-Import-Interface

- CSV-Datei: „comma-separated values“  
Tabelle, durch Kommata getrennt
- Viele Tabellenkalkulationsprogramme können CSV-Dateien exportieren.
- Beispiel-Datei: [test.csv](#) (siehe auch: [test.ods](#))
- Beispiel-Anwendung: Ermitteln der durchschnittlichen Note
- Beispiel-Anwendung für Datenbank-Experten:  
Speichere die Spalten 1 und 2 in einer Datenbank.
- Hinweis 1: `cat X-1.txt`
- Hinweis 2: `man cut`

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in 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

2.7 Skript-Programmierung

## 3 TCP/IP in der Praxis

...

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

24. November 2020

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in 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

### 2.7 Skript-Programmierung

### 2.8 GNU screen

## 3 TCP/IP in der Praxis

## 4 Bus-Systeme

...

## 2.4 Ein- und Ausgabeströme

- Standard-Ausgabe in Datei umleiten

```
$ echo "Dies ist ein Test." > test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

- Standard-Ausgabe an Datei anhängen

```
$ echo "Dies ist noch ein Test." >> test.txt
```

```
$ cat test.txt
```

```
Dies ist ein Test.
```

```
Dies ist noch ein Test.
```

## 2.4 Ein- und Ausgabeströme

- Fehler-Ausgabe in Datei umleiten

```
$ cat gibtsnicht.txt > fehler.txt
cat: gibtsnicht.txt: No such file or directory
$ cat fehler.txt
$ cat gibtsnicht.txt 2> fehler.txt
$ cat fehler.txt
cat: gibtsnicht.txt: No such file or directory
```

## 2.4 Ein- und Ausgabeströme

- Standard-Eingabe aus Datei lesen

```
$ bc
bc 1.06.95
Copyright [...] 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
2 + 2
4
$ echo "2 + 2" > test.bc
$ bc < test.bc
4
```



## 2.5 Pipes

Standard-Ausgabe von Programm A  
wird zu Standard-Eingabe von Programm B

```
$ echo "2 + 2" | bc  
4
```

→ sehr mächtiger „Baukasten“

## 2.5 Pipes

- `sed`: *stream editor*

Suchen und Ersetzen (und noch viel mehr)

```
$ echo "Schlimmer geht nimmer." | sed -e 's/nim/im/g'  
Schlimmer geht immer.
```

## 2.5 Pipes

- **grep**: Standard-Eingabe durchsuchen

```
$ ls | grep slides
```

```
pgslides.sty
```

```
$ ls *.pdf | grep -v logo
```

```
es-20191002.pdf
```

```
Zeichen_123.pdf
```

```
$ ls -l $(ls *.pdf | grep -v logo)
```

```
-rw-r--r-- 1 ... 4619138 Okt 8 21:28 es-20191002.pdf
```

```
lrwxrwxrwx 1 ...          25 Okt 3   2016 Zeichen_123.pdf -> ...
```

## 2.5 Pipes

- Standard-Ausgabe im Befehl verwenden

```
$ grep -l "|" *.txt
regex-1.txt
regex-2.txt
regex-3.txt
regex-4.txt
shell-skripts-11.txt
X-1.txt
$ vi $(grep -l "|" *.txt)
```

## 2.6 Verzweigungen und Schleifen

```
$ if grep Blubb test.txt; then echo "gefunden"; \  
  else echo "nicht gefunden"; fi  
nicht gefunden  
$ for x in foo bar baz; do echo $x; done  
foo  
bar  
baz
```

## 2.7 Skript-Programmierung

**Aufgabe:** Programmieren Sie ein Unix-Shell-Skript, das etwas Nützliches macht. :-)

Beispiel: CSV-Import-Interface

- CSV-Datei: „comma-separated values“  
Tabelle, durch Kommata getrennt
- Viele Tabellenkalkulationsprogramme können CSV-Dateien exportieren.
- Beispiel-Datei: [test.csv](#) (siehe auch: [test.ods](#))
- Beispiel-Anwendung: Ermitteln der durchschnittlichen Note
- Beispiel-Anwendung für Datenbank-Experten:  
Speichere die Spalten 1 und 2 in einer Datenbank.
- Hinweis 1: `cat X-1.txt`
- Hinweis 2: `man cut`

## 2.8 GNU screen

- Text-Bildschirm und Eingabegeräte über's Netz
- `Ctrl+A c`: neues Fenster (create)
- `Ctrl+A Leertaste`: nächstes Fenster
- `Ctrl+A 3`: Fenster Nr. 3
- `Ctrl+A ESC`: hochscrollen, suchen, markieren (copy)
- `Ctrl+A ]`: einfügen (paste)
- `Ctrl+A d`: von `screen` ablösen (detach)
- `screen -x`: an laufenden `screen` andocken
- ähnliche Funktionalität für Grafik: `VNC`, `x2go`

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in 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

### 2.7 Skript-Programmierung

### 2.8 GNU screen

## 3 TCP/IP in der Praxis

## 4 Bus-Systeme

...



# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
  - 3.0** Vorbereitungen
  - 3.1** IP-Adressen
  - 3.2** MAC-Adressen
  - 3.3** TCP- und UDP-Ports
  - 3.4** TCP-Protokolle
  - 3.5** Routing
  - 3.6** Netzwerkanalyse
  - 3.7** SSH
  - 3.8** X11
  - 3.9** Programmierung
- 4 Bus-Systeme**

...

# 3 TCP/IP in der Praxis

## Literatur:

<http://www.peter.gerwinski.de/download/net-2013ss.tar.gz>

## 3.0 Vorbereitungen **(Nicht jetzt ausführen!)**

- Verkabelung: Twisted-Pair-Kabel, Switches
- Automatismen abschalten

```
# service network-manager stop
```

## 3.1 IP-Adressen

- `ip addr` (Linux)
- `ifconfig` (Unix allgemein)
- `ipconfig` (MS Windows)
- `ip addr add <Netz>`
- `ip link`
- `ping <IP-Adresse>`

```
# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    [...]

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.42.101  netmask 255.255.255.0
                                broadcast 192.168.42.255
    ether be:3f:ca:aa:7e:51 txqueuelen 1000  (Ethernet)
    [...]
```

## 3.1 IP-Adressen

IPv4-Adressen:

- 32 Bit
- dezimal, 4 Gruppen zu je 8 Bit (0–255), durch Punkte getrennt

IPv6-Adressen:

- 128 Bit
- hexadezimal, 8 Gruppen zu je 4 Hex-Ziffern, durch Doppelpunkte getrennt
- Führende Nullen dürfen weggelassen werden.
- Zwei Doppelpunkte bedeuten: Mit Nullen auffüllen.
- Literatur und Beispiel: <https://de.wikipedia.org/wiki/IPv6>

## 3.2 MAC-Adressen

MAC = Media Access Control

MAC-Adresse = Hardware-Adresse = Ethernet-Adresse

- `ip neigh`  
`arp`

## 3.3 TCP- und UDP-Ports

- `nc <IP> <Port>`  
Verbindung zu Programm  $\langle$ Port $\rangle$  auf Rechner  $\langle$ IP $\rangle$  aufnehmen
- `nc -l <Port>` oder `nc -p <Port> -l`  
auf eingehende Verbindungen warten („lauschen“)
- TCP-Ports: Verbindungskonzept, Netzwerk prüft
- UDP-Ports: einzelne Pakete, Anwendung muß selbst prüfen
- ICMP: keine Ports, nur Rechner:  
Erreichbarkeit, Eigenschaften der Übertragung

Anwendung: HTTP, SMTP, ...
Transport: TCP-/UDP-Ports, ICMP
Internet: IP-Adresse
Netzwerkzugang: Hardware-/MAC-Adresse

## 3.4 TCP-Protokolle

- **HTTP**

GET / HTTP/1.1

Host: www.hs-bochum.de

(Leerzeile)

## 3.4 TCP-Protokolle

- **HTTP**

```
GET / HTTP/1.1  
Host: www.hs-bochum.de  
(Leerzeile)
```

- **SMTP**

```
HELO cassini  
MAIL FROM: <example@example.com>  
RCPT TO: <beispiel@example.de>  
(E-Mail-Header – Teil der Nutzdaten)  
From: Eddie Example <example@example.com>  
To: Bert Beispiel <beispiel@example.de>  
Subject: Hello, world!  
(Leerzeile)  
Hi, there!  
.
```

- Protokolle „mal eben“ selbst schreiben: `nc -c` oder `inetsd`



# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

1. Dezember 2020

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
  - 3.0 Vorbereitungen
  - 3.1 IP-Adressen
  - 3.2 MAC-Adressen
  - 3.3 TCP- und UDP-Ports
  - 3.4 TCP-Protokolle
  - 3.5 Routing
  - 3.6 Netzwerkanalyse
  - 3.7 SSH
  - 3.8 X11
  - 3.9 Programmierung
- 4 Bus-Systeme**

...

# 3 TCP/IP in der Praxis

## Literatur:

<http://www.peter.gerwinski.de/download/net-2013ss.tar.gz>

## 3.0 Vorbereitungen **(Nicht jetzt ausführen!)**

- Verkabelung: Twisted-Pair-Kabel, Switches
- Automatismen abschalten

```
# service network-manager stop
```

## 3.1 IP-Adressen

- `ip addr` (Linux)
- `ifconfig` (Unix allgemein)
- `ipconfig` (MS Windows)
- `ip addr add <Netz>`
- `ip link`
- `ping <IP-Adresse>`

```
# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    [...]

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.42.101  netmask 255.255.255.0
                                broadcast 192.168.42.255
    ether be:3f:ca:aa:7e:51 txqueuelen 1000  (Ethernet)
    [...]
```

## 3.1 IP-Adressen

IPv4-Adressen:

- 32 Bit
- dezimal, 4 Gruppen zu je 8 Bit (0–255), durch Punkte getrennt

IPv6-Adressen:

- 128 Bit
- hexadezimal, 8 Gruppen zu je 4 Hex-Ziffern, durch Doppelpunkte getrennt
- Führende Nullen dürfen weggelassen werden.
- Zwei Doppelpunkte bedeuten: Mit Nullen auffüllen.
- Literatur und Beispiel: <https://de.wikipedia.org/wiki/IPv6>

## 3.2 MAC-Adressen

MAC = Media Access Control

MAC-Adresse = Hardware-Adresse = Ethernet-Adresse

- `ip neigh`  
`arp`

## 3.3 TCP- und UDP-Ports

- `nc <IP> <Port>`  
Verbindung zu Programm  $\langle$ Port $\rangle$  auf Rechner  $\langle$ IP $\rangle$  aufnehmen
- `nc -l <Port>` oder `nc -p <Port> -l`  
auf eingehende Verbindungen warten („lauschen“)
- TCP-Ports: Verbindungskonzept, Netzwerk prüft
- UDP-Ports: einzelne Pakete, Anwendung muß selbst prüfen
- ICMP: keine Ports, nur Rechner:  
Erreichbarkeit, Eigenschaften der Übertragung

Anwendung: HTTP, SMTP, ...
Transport: TCP-/UDP-Ports, ICMP
Internet: IP-Adresse
Netzwerkzugang: Hardware-/MAC-Adresse

## 3.4 TCP-Protokolle

- **HTTP**

GET / HTTP/1.1

Host: www.hs-bochum.de

(Leerzeile)

URL: Schema://Benutzer:Passwort@Rechner:port/Pfad?Query#Fragment



## 3.4 TCP-Protokolle

- **HTTP**

- **SMTP**

HELO cassini

MAIL FROM: <example@example.com>

RCPT TO: <beispiel@example.de>

(E-Mail-Header – Teil der Nutzdaten)

DATA

From: Eddie Example <example@example.com>

To: Bert Beispiel <beispiel@example.de>

Subject: Hello, world!

(Leerzeile)

Hi, there!

.

- Protokolle „mal eben“ selbst schreiben: `nc -c` oder `inetd`

## 3.5 Routing

- `ip route` (Linux)  
`route` (MS-Windows, Unix)  
`netstat -nr` (MacOS)

```
# route -n
```

```
Kernel-IP-Routentabelle
```

Ziel	Router	Genmask	[...]	Iface
0.0.0.0	192.168.42.1	0.0.0.0	[...]	wlan0
169.254.0.0	0.0.0.0	255.255.0.0	[...]	wlan0
192.168.42.0	0.0.0.0	255.255.255.0	[...]	wlan0

Netzmaske:

Wenn nach Und-Verknüpfung mit IP-Adresse gleich, → im gleichen Netz

255.255.240.0 ist dasselbe wie /20

(20 Bit sind 1; die restlichen 12 Bit sind 0)

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16
- (d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

Die Netzmaske /24 steht für 24 Einsen, 8 Nullen,  
ausgeschrieben 11111111 11111111 11111111 00000000  
oder 255.255.255.0

→ Die ersten drei Zahlen bezeichnen das Netz,  
die letzte Zahl den Rechner im Netz.

Da sich die ersten drei Zahlen für diese beiden Rechner unterscheiden,  
befinden sie sich in unterschiedlichen Netzen.

(b) bei Netzmaske /20

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16

Die Netzmaske /16 steht für 16 Einsen, 16 Nullen,  
ausgeschrieben 11111111 11111111 00000000 00000000  
oder 255.255.0.0

→ Die ersten zwei Zahlen bezeichnen das Netz,  
die letzten zwei Zahlen den Rechner im Netz.

Da die ersten zwei Zahlen für diese beiden Rechner gleich sind,  
befinden sie sich im gleichen Netz.

- (d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16

Ich mache eine Und-Verknüpfung von 10.11.12.13 mit 255.255.0.0 und erhalte 10.11.0.0.

Ich mache eine Und-Verknüpfung von 10.11.22.33 mit 255.255.0.0 und erhalte ebenfalls 10.11.0.0.

Da beide Ergebnisse gleich sind, befinden sich beide Rechner im gleichen Netz.

- (d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

(b) bei Netzmaske /20

Die Netzmaske /20 ist ausgeschrieben

11111111 11111111 11110000 00000000. Ich mache eine Und-Verknüpfung

von 10.11.12.13 = 00001010 00001011 00001100 00001101, mit der

Netzmaske und erhalte 00001010 00001011 00000000 00000000 =

10.11.0.0. Ich mache eine Und-Verknüpfung von 10.11.22.33 =

00001010 00001011 00010110 00010001, und erhalte

00001010 00001011 00010000 00000000 = 10.11.16.0.

Da beide Ergebnisse verschieden sind,

befinden sich beide Rechner nicht im gleichen Netz.

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält,

d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

(b) bei Netzmaske /20

Die Netzmaske /20 ist ausgeschrieben 255.255.255.240.

Ich mache eine Und-Verknüpfung von 10.11.12.13 mit 255.255.240.0 und erhalte 10.11.0.0.

Ich mache eine Und-Verknüpfung von 10.11.22.33 mit 255.255.240.0 und erhalte ebenfalls 10.11.16.0.

Da beide Ergebnisse verschieden sind, befinden sich beide Rechner nicht im gleichen Netz.

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist.



## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16
- (d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist. Wir notieren beide IP-Adressen binär und schauen, ab dem wievielten Bit sie sich unterscheiden:

10.11.12.13 = 00001010 00001011 00001100 00001101

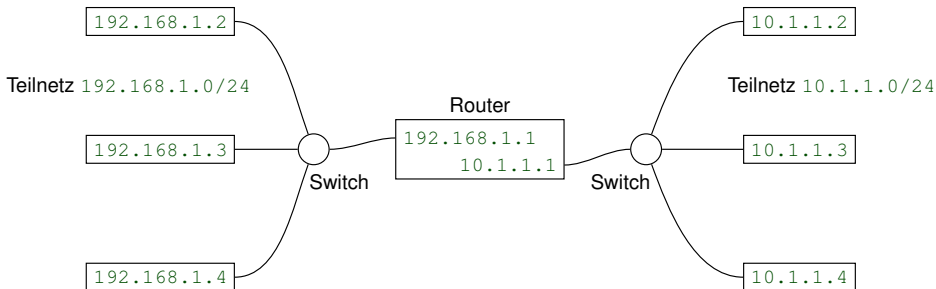
10.11.22.33 = 00001010 00001011 00010110 00010001

Die IP-Adressen stimmen in den ersten 19 Bits überein.

Die Maske des kleinsten Netzes, das beide Rechner enthält, lautet daher:

255.255.224.0 = 11111111 11111111 11100000 00000000 = /19.

## 3.5 Routing



Private IPv4-Netze:

10.0.0.0/8

172.16.0.0/12 – private: werden im öffentlichen Internet nicht geroutet

192.168.0.0/16

169.254.0.0/16 – link local: überhaupt kein Routing vorgesehen

## 3.5 Routing

- Private IP-Adressen:  
kein Routing im öffentlichen Internet

- *Network Address Translation (NAT, IP-Masquerading)*  
Merken der ursprünglichen IP über die Ausgangs-Port-Nr.
- Probleme mit Protokollen, die das Schichtenmodell verletzen,  
z. B. FTP, aber auch WebRTC
- zusätzliche Maßnahmen, z. B. STUN- oder TURN-Server

- Firewall-Regel:

```
iptables -t nat -A POSTROUTING \  
        -o WAN-Interface -j MASQUERADE
```

## 3.6 Netzwerkanalyse

- tcpdump
- wireshark
- ettercap

**Warnung:** Das unerlaubte Mitlesen oder Manipulieren von Netzwerkverkehr ist gemäß deutschem Recht eine Straftat, die eine mehrjährige Freiheitsstrafe nach sich ziehen kann.

Bei der Anwendung der hier vorgestellten Werkzeuge ist daher höchste Sorgfalt geboten, um nicht versehentlich die Grenze zur Kriminalität zu überschreiten. Die Situation ist vergleichbar mit der Steuerung schwerer Maschinen, die bei unsachgemäßer Handhabung erheblichen Sach- und/oder Personenschaden bewirken können.

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

8. Dezember 2020

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in Unix

## 3 TCP/IP in der Praxis

### 3.0 Vorbereitungen

### 3.1 IP-Adressen

### 3.2 MAC-Adressen

### 3.3 TCP- und UDP-Ports

### 3.4 TCP-Protokolle

### 3.5 Routing

### 3.6 Netzwerkanalyse

### 3.7 SSH

### 3.8 X11 und VNC

### 3.9 Programmierung

## 4 Bus-Systeme

...

## 3.5 Routing

- `ip route` (Linux)  
`route` (MS-Windows, Unix)  
`netstat -nr` (MacOS)

```
# route -n
```

```
Kernel-IP-Routentabelle
```

Ziel	Router	Genmask	[...]	Iface
0.0.0.0	192.168.42.1	0.0.0.0	[...]	wlan0
169.254.0.0	0.0.0.0	255.255.0.0	[...]	wlan0
192.168.42.0	0.0.0.0	255.255.255.0	[...]	wlan0

Netzmaske:

Wenn nach Und-Verknüpfung mit IP-Adresse gleich, → im gleichen Netz

255.255.240.0 ist dasselbe wie /20

(20 Bit sind 1; die restlichen 12 Bit sind 0)

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16
- (d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist.



## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

Die Netzmaske /24 steht für 24 Einsen, 8 Nullen,  
ausgeschrieben 11111111 11111111 11111111 00000000  
oder 255.255.255.0

→ Die ersten drei Zahlen bezeichnen das Netz,  
die letzte Zahl den Rechner im Netz.

Da sich die ersten drei Zahlen für diese beiden Rechner unterscheiden,  
befinden sie sich in unterschiedlichen Netzen.

(b) bei Netzmaske /20

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16

Die Netzmaske /16 steht für 16 Einsen, 16 Nullen,  
ausgeschrieben 11111111 11111111 00000000 00000000  
oder 255.255.0.0

→ Die ersten zwei Zahlen bezeichnen das Netz,  
die letzten zwei Zahlen den Rechner im Netz.

Da die ersten zwei Zahlen für diese beiden Rechner gleich sind,  
befinden sie sich im gleichen Netz.

- (d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16

Ich mache eine Und-Verknüpfung von 10.11.12.13 mit 255.255.0.0 und erhalte 10.11.0.0.

Ich mache eine Und-Verknüpfung von 10.11.22.33 mit 255.255.0.0 und erhalte ebenfalls 10.11.0.0.

Da beide Ergebnisse gleich sind, befinden sich beide Rechner im gleichen Netz.

- (d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

(b) bei Netzmaske /20

Die Netzmaske /20 ist ausgeschrieben

11111111 11111111 11110000 00000000. Ich mache eine Und-Verknüpfung

von 10.11.12.13 = 00001010 00001011 00001100 00001101, mit der

Netzmaske und erhalte 00001010 00001011 00000000 00000000 =

10.11.0.0. Ich mache eine Und-Verknüpfung von 10.11.22.33 =

00001010 00001011 00010110 00010001, und erhalte

00001010 00001011 00010000 00000000 = 10.11.16.0.

Ergebnisse verschieden → nicht im gleichen Netz.

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält,  
d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält,  
dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

(a) bei Netzmaske /24

(b) bei Netzmaske /20

Die Netzmaske /20 ist ausgeschrieben 255.255.255.240.

Ich mache eine Und-Verknüpfung von 10.11.12.13 mit 255.255.240.0 und erhalte 10.11.0.0.

Ich mache eine Und-Verknüpfung von 10.11.22.33 mit 255.255.240.0 und erhalte ebenfalls 10.11.16.0.

Da beide Ergebnisse verschieden sind, befinden sich beide Rechner nicht im gleichen Netz.

(c) bei Netzmaske /16

(d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist.

## 3.5 Routing

### Aufgabe

Zwei Rechner haben die IP-Adressen 10.11.12.13 und 10.11.22.33.

Befinden sich die beiden Rechner im gleichen Netz?

- (a) bei Netzmaske /24
- (b) bei Netzmaske /20
- (c) bei Netzmaske /16
- (d) Bestimme das kleinste Netz, das beide Rechner enthält, d. h. dasjenige Netz, das die wenigsten IP-Adressen enthält, dessen Zahl hinter dem Schrägstrich also möglichst groß ist. Wir notieren beide IP-Adressen binär und schauen, ab dem wievielten Bit sie sich unterscheiden:

10.11.12.13 = 00001010 00001011 00001100 00001101

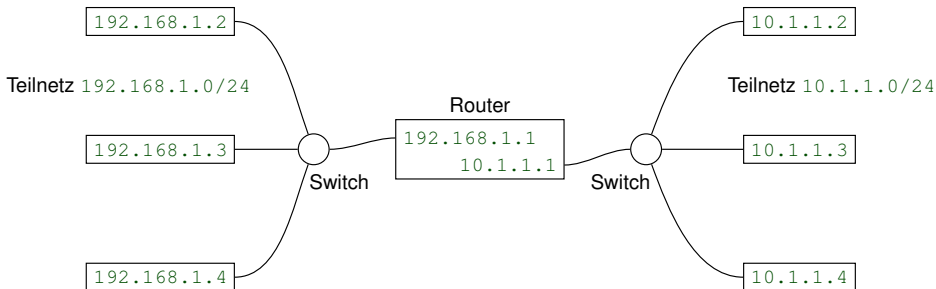
10.11.22.33 = 00001010 00001011 00010110 00010001

Die IP-Adressen stimmen in den ersten 19 Bits überein.

Die Maske des kleinsten Netzes, das beide Rechner enthält, lautet daher:

255.255.224.0 = 11111111 11111111 11100000 00000000 = /19.

## 3.5 Routing



Private IPv4-Netze:

10.0.0.0/8

172.16.0.0/12 – private: werden im öffentlichen Internet nicht geroutet

192.168.0.0/16

169.254.0.0/16 – link local: überhaupt kein Routing vorgesehen

## 3.5 Routing

- Private IP-Adressen:  
kein Routing im öffentlichen Internet

- *Network Address Translation (NAT, IP-Masquerading)*  
Merken der ursprünglichen IP über die Ausgangs-Port-Nr.
- Probleme mit Protokollen, die das Schichtenmodell verletzen,  
z. B. FTP, aber auch WebRTC
- zusätzliche Maßnahmen, z. B. STUN- oder TURN-Server

- Routing einschalten:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Firewall-Regel:

```
iptables -t nat -A POSTROUTING \  
    -o WAN-Interface -j MASQUERADE
```



## 3.6 Netzwerkanalyse

- tcpdump
- wireshark
- ettercap

**Warnung:** Das unerlaubte Mitlesen oder Manipulieren von Netzwerkverkehr ist gemäß deutschem Recht eine Straftat, die eine mehrjährige Freiheitsstrafe nach sich ziehen kann.

Bei der Anwendung der hier vorgestellten Werkzeuge ist daher höchste Sorgfalt geboten, um nicht versehentlich die Grenze zur Kriminalität zu überschreiten. Die Situation ist vergleichbar mit der Steuerung schwerer Maschinen, die bei unsachgemäßer Handhabung erheblichen Sach- und/oder Personenschaden bewirken können.

## 3.7 SSH

- Mit `nc -p 2345 -l -c /bin/bash` ermögliche ich Shell-Zugriff auf den Rechner über Port 2345.
- Achtung: Es erfolgt keine Verschlüsselung und noch nicht einmal eine Passwort-Abfrage!

Richtige Lösung mit Verschlüsselung und Authentifizierung: SSH

- `SSH <Rechner>`
- `-C`: Komprimierung
- `-L`: lokalen Port auf Remote-Port umleiten
- `-R`: Remote-Port auf lokalen Port umleiten

Authentifizierung

- Ich hinterlege den öffentlichen Schlüssel auf dem Ziel-Rechner.
- Wer dann im Besitz des privaten Schlüssels ist, kann sich dann auf dem Ziel-Rechner einloggen.
- Diese Methode ist sicherer als die Abfrage eines Passworts.

## 3.8 X11 und VNC

- Grafik-Bildschirm und Eingabegeräte über's Netz
- `DISPLAY`-Variable: X-Server: Rechner und Bildschirm
- `ssh -X`: X11-Forwarding

## 3.8 X11 und VNC

- Grafik-Bildschirm und Eingabegeräte über's Netz
- `DISPLAY`-Variable: X-Server: Rechner und Bildschirm
- `ssh -X`: X11-Forwarding
  
- VNC = Virtual Network Computing
- VNC-Server stellt Bildschirminhalt zur Verfügung
  - entweder: eigener, virtueller X11-Server
  - oder: ruft Inhalt von anderem (X11-) Bildschirm ab
- VNC-Client ruft Bildschirminhalt ab und stellt ihn dar
  - z. B. per X11
  - z. B. per Web-Interface: noVNC

## 3.9 Programmierung

- Shell-Skripte: `nc` (`traditional` oder `OpenBSD`)
- C: Sockets, `select()`
- C++: Callbacks

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

15. Dezember 2020

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in Unix

## 3 TCP/IP in der Praxis

### 3.0 Vorbereitungen

### 3.1 IP-Adressen

### 3.2 MAC-Adressen

### 3.3 TCP- und UDP-Ports

### 3.4 TCP-Protokolle

### 3.5 Routing

### 3.6 Netzwerkanalyse

### 3.7 SSH

### 3.8 X11 und VNC

### 3.9 Programmierung

## 4 Versionsverwaltungssysteme

## 5 Bus-Systeme

...

## 3.7 SSH

- Mit `nc -p 2345 -l -c /bin/bash` ermögliche ich Shell-Zugriff auf den Rechner über Port 2345.
- Achtung: Es erfolgt keine Verschlüsselung und noch nicht einmal eine Passwort-Abfrage!

Richtige Lösung mit Verschlüsselung und Authentifizierung: SSH

- `SSH <Rechner>`
- `-C`: Komprimierung
- `-L`: lokalen Port auf Remote-Port umleiten
- `-R`: Remote-Port auf lokalen Port umleiten

Authentifizierung

- Ich hinterlege den öffentlichen Schlüssel auf dem Ziel-Rechner.
- Wer dann im Besitz des privaten Schlüssels ist, kann sich dann auf dem Ziel-Rechner einloggen.
- Diese Methode ist sicherer als die Abfrage eines Passworts.



## 3.8 X11 und VNC

- Grafik-Bildschirm und Eingabegeräte über's Netz
- `DISPLAY`-Variable: X-Server: Rechner und Bildschirm
- `ssh -X`: X11-Forwarding

## 3.8 X11 und VNC

- Grafik-Bildschirm und Eingabegeräte über's Netz
- `DISPLAY`-Variable: X-Server: Rechner und Bildschirm
- `ssh -X`: X11-Forwarding
  
- VNC = Virtual Network Computing
- VNC-Server stellt Bildschirminhalt zur Verfügung
  - entweder: eigener, virtueller X11-Server
  - oder: ruft Inhalt von anderem (X11-) Bildschirm ab
- VNC-Client ruft Bildschirminhalt ab und stellt ihn dar
  - z. B. per X11
  - z. B. per Web-Interface: noVNC

## 3.9 Programmierung

- Shell-Skripte: nc (traditional oder OpenBSD)
- C: Sockets, select()
- C++: Callbacks

## 3.9 Programmierung

- Shell-Skripte: `nc` (`traditional` oder `OpenBSD`)
- C: Sockets, `select()`
- C++: Callbacks

Aktuelles Beispiel: Programmierung eines VNC-Servers per Web-Interface

- Grundlagen: RFC 6143
- Praxis: Untersuchung mit Wireshark
- Implementation: JavaScript, WebSockets, `websocketify`, Callbacks

# 4 Versionsverwaltungssysteme

## 4.1 Historisches

Bekannte Versionsverwaltungswerkzeuge:

**1974** diff

**1982** RCS – Revision Control System

**1985** patch

**1990** CVS – Concurrent Versions System

**2000** SVN – Subversion

**2005** Git

**2005** Mercurial

... und *vielen* weitere!

# 4 Versionsverwaltungssysteme

## 4.2 Git

- Name: „git“ (engl.) = „Idiot“
- Initiator: Linus Torvalds  
*„I'm an egotistical bastard, and I name all my projects after myself.  
First ,Linux', now ,Git'.“*

# 4 Versionsverwaltungssysteme

## 4.2 Git

- Name: „git“ (engl.) = „Idiot“
- Initiator: Linus Torvalds  
*„I'm an egotistical bastard, and I name all my projects after myself. First ,Linux', now ,Git'.“*
- **Repository** – „Behälter“, in dem alle Dateien liegen  
– einschließlich sämtlicher historischer Versionsn
- bei Git: dezentral
- **Branches** – parallele Versionen desselben Repository
- bei Git: leistungsfähige Werkzeuge, um Branches wieder zusammenzuführen

# 4 Versionsverwaltungssysteme

## 4.2 Git

- *commit* – Änderung in das Repository übernehmen
- *stage* – für *commit* vorgemerkte Änderungen
- Unterverzeichnis *.git*: Konfiguration und Repository

Befehle:

- *git status* – Status anzeigen  
Welche Dateien sind in der *stage*? Welche sind neu?
- *git add* – Änderungen vormerken
- *git commit* – Änderungen in Repository übernehmen
- *git checkout -- <Datei>* – Datei aus Repository zurückholen
- *git push* – Änderungen in anderes Repository übertragen
- *git clean* – temporäre Dateien löschen
- *git init* – Verzeichnis für Arbeit mit Git vorbereiten



# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

22. Dezember 2020

# Eingebettete Systeme

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

## 1 Einführung

## 2 Einführung in Unix

## 3 TCP/IP in der Praxis

### 3.8 X11 und VNC

### 3.9 Programmierung

## 4 Versionsverwaltungssysteme

### 4.1 Historisches

### 4.2 diff und patch

### 4.3 Git

## 5 Bus-Systeme

### 5.1 Was sind Bus-Systeme?

### 5.2 RS-232

### 5.3 I<sup>2</sup>C (TWI)

### 5.4 SPI

...

...

## 3 TCP/IP in der Praxis

### 3.8 X11 und VNC

- Grafik-Bildschirm und Eingabegeräte über's Netz
- `DISPLAY`-Variable: X-Server: Rechner und Bildschirm
- `ssh -X`: X11-Forwarding
  
- VNC = Virtual Network Computing
- VNC-Server stellt Bildschirminhalt zur Verfügung
  - entweder: eigener, virtueller X11-Server
  - oder: ruft Inhalt von anderem (X11-) Bildschirm ab
- VNC-Client ruft Bildschirminhalt ab und stellt ihn dar
  - z. B. per X11
  - z. B. per Web-Interface: noVNC

## 3 TCP/IP in der Praxis

### 3.9 Programmierung

- Shell-Skripte: `nc` (`traditional` oder `OpenBSD`)
- C: Sockets, `select()`
- C++: Callbacks

Aktuelles Beispiel: Programmierung eines VNC-Servers per Web-Interface

- Grundlagen: RFC 6143
- Praxis: Untersuchung mit Wireshark
- Implementation: JavaScript, WebSockets, `websocketify`, Callbacks

# 4 Versionsverwaltungssysteme

## 4.1 Historisches

Bekannte Versionsverwaltungswerkzeuge:

**1974** diff

**1982** RCS – Revision Control System

**1985** patch

**1990** CVS – Concurrent Versions System

**2000** SVN – Subversion

**2005** Git

**2005** Mercurial

... und *vielen* weitere!

# 4 Versionsverwaltungssysteme

## 4.2 diff und patch

- Standard-Werkzeuge unter Unix
- `diff`: Textdateien miteinander vergleichen
- `diff -w` – Leerzeichen und Tabulatoren (whitespace) ignorieren
- `diff -c` – Kontext (context) anzeigen
- `diff -u` – Gemeinsamen (unified) Kontext anzeigen
- `diff -p` – Funktion (procedure) anzeigen
- `patch`: In `diff`-Datei gespeicherte Änderungen automatisch ausführen
- Dadurch möglich: parallele Änderungen an einer Datei (branches) automatisch zusammenführen (merge)
- Bei Konflikten entsteht eine neue `diff`-Datei mit der Endung `rej` (reject).

## 4 Versionsverwaltungssysteme

### 4.3 Git

- Name: „git“ (engl.) = „Idiot“
- Initiator: Linus Torvalds  
*„I'm an egotistical bastard, and I name all my projects after myself. First ,Linux', now ,Git'.“*
- **Repository** – „Behälter“, in dem alle Dateien liegen  
– einschließlich sämtlicher historischer Versionsn
- bei Git: dezentral
- **Branches** – parallele Versionen desselben Repository
- bei Git: leistungsfähige Werkzeuge, um Branches wieder zusammenzuführen

# 4 Versionsverwaltungssysteme

## 4.3 Git

- *commit* – Änderung in das Repository übernehmen
- *stage* – für *commit* vorgemerkte Änderungen
- Unterverzeichnis *.git*: Konfiguration und Repository

Befehle:

- *git status* – Status anzeigen  
Welche Dateien sind in der *stage*? Welche sind neu?
- *git add* – Änderungen vormerken
- *git commit* – Änderungen in Repository übernehmen
- *git checkout -- <Datei>* – Datei aus Repository zurückholen
- *git push* – Änderungen in anderes Repository übertragen
- *git clean* – temporäre Dateien löschen
- *git init* – Verzeichnis für Arbeit mit Git vorbereiten



# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

*Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg.*

[https://de.wikipedia.org/wiki/Bus\\_\(Datenverarbeitung\)](https://de.wikipedia.org/wiki/Bus_(Datenverarbeitung))

Beispiele:

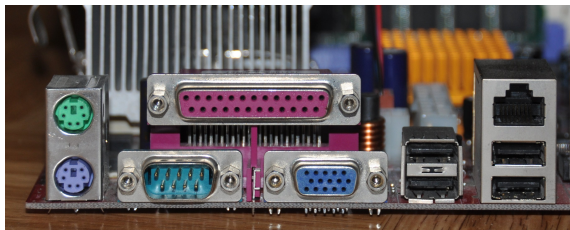
- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

Standard-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics



# 5 Bus-Systeme

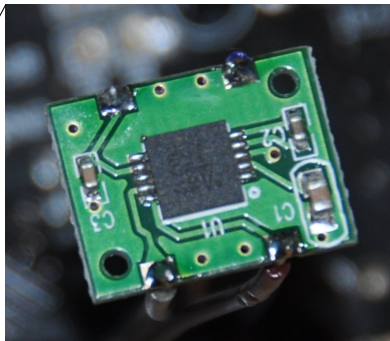
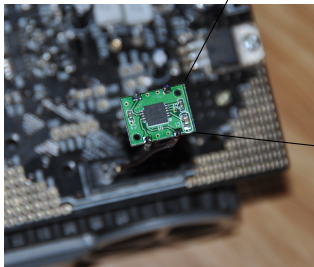
## 5.1 Was sind Bus-Systeme?

Standard-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics

Minimal-Hardware:

- RS-232
- I<sup>2</sup>C (TWI)
- SPI



# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

<i>seriell</i>	jedes Bit einzeln übertragen
<i>parallel</i>	mehrere Bits gleichzeitig
<i>synchron</i>	Abgleich über Steuerleitung: <i>Takt</i>
<i>asynchron</i>	Abgleich über Zeitvereinbarungen
<i>Punkt-zu-Punkt</i>	genau zwei Teilnehmer
<i>busfähig</i>	mehrere Teilnehmer, mit <i>Adressierung</i>

- I<sup>2</sup>C: seriell, synchron, mit Adressierung
- RS-232: seriell, asynchron, Punkt-zu-Punkt
- RS-485, USB, CAN: seriell, asynchron, mit Adressierung
- SPI: seriell, synchron, Punkt-zu-Punkt oder mit Adressierung

# 5 Bus-Systeme

## 5.2 RS-232

seriell

- *TX*: 1 Leitung für Daten
- *RX*: ggf. 1 Leitung für Daten in der anderen Richtung
- *GND*: gemeinsame *Masse*
- evtl. zusätzliche Steuerleitungen

asynchron

- *keine* Taktleitung für Abgleich, wann Daten anliegen
- Stattdessen: Abgleich über Zeitvereinbarungen

→ Jeder Teilnehmer braucht eine eigene Zeitbasis.

Punkt-zu-Punkt

- nur 2 Teilnehmer vorgesehen

## 5.2 RS-232

Synchronisation

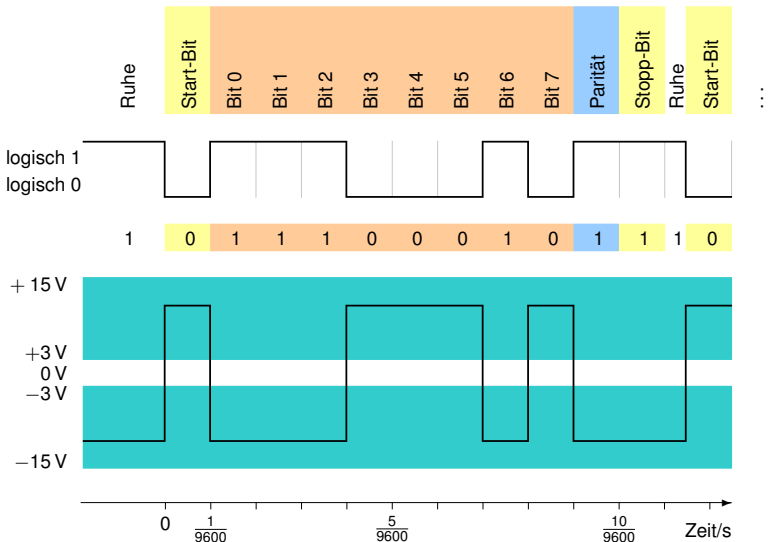
Daten

Check

9600 Baud, 8 Daten-Bits, ungerade Parität, 1 Stopp-Bit

Beispiel-Daten: ASCII „G“ = 71 = 0100 0111 binär

Übertragung der Daten von rechts (Bit 0) nach links (Bit 7)



# 5 Bus-Systeme

## 5.3 I<sup>2</sup>C (TWI)

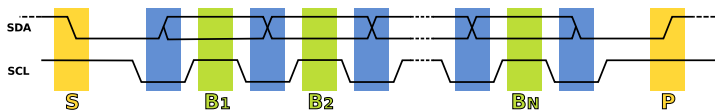
I<sup>2</sup>C = Inter-IC; TWI = Two-Wire-Interface

seriell

- *SDA*: 1 Leitung für Daten (in beiden Richtungen)
- *SCL*: Taktleitung (Clock)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung



busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- erstes gesendetes Byte: *Adresse* des Teilnehmers
- 2 Adressen pro Teilnehmer: Lesen/Schreiben

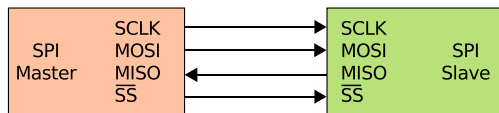
# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät



synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

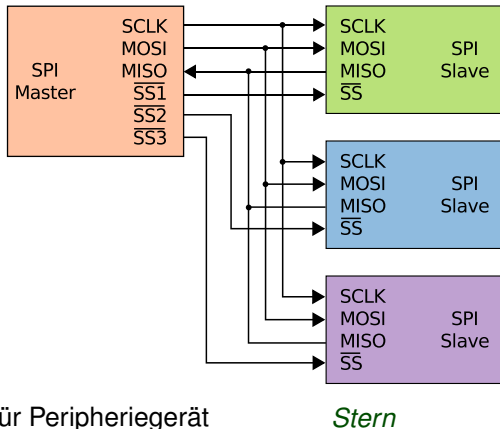
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

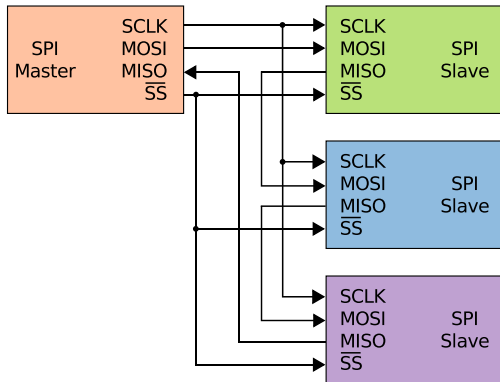
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



*Kaskade  
Daisy Chain*

# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

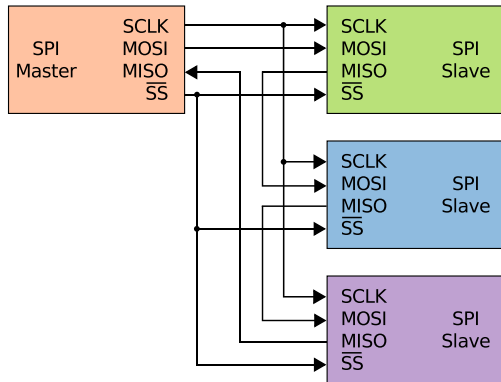
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



*Kaskade  
Daisy Chain*

Slave gibt MOSI-Input um 1 Takt verzögert an MISO aus → Master setzt „im richtigen Moment“  $\overline{SS}$

# 5 Bus-Systeme

## 5.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

- Steuerung von Motoren
- Nutzung als allgemeines Protokoll zur Übertragung analoger Werte

# 5 Bus-Systeme

## 5.6 Sonstiges

### Matrix-Schaltung

- möglichst viele Aktoren/Sensoren  
über möglichst wenige digital Inputs abfragen  
bzw. über möglichst wenige digitale Outputs steuern
- Beispiele: LED-Felder, Tastaturen

### R/2R-Netzwerk

- möglichst viele digitale Inputs  
über einen einzigen analogen Input abfragen
- Beispiele: Tastaturen

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

5. Januar 2021

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
  - 5.1 Was sind Bus-Systeme?
  - 5.2 RS-232
  - 5.3 I<sup>2</sup>C (TWI)
  - 5.4 SPI
  - 5.5 PWM
  - 5.6 Sonstiges
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?
  - 6.2 Echtzeitprogrammierung
  - 6.3 Multitasking
  - 6.4 Ressourcen
  - 6.5 Prioritäten

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

*Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg.*

[https://de.wikipedia.org/wiki/Bus\\_\(Datenverarbeitung\)](https://de.wikipedia.org/wiki/Bus_(Datenverarbeitung))

Beispiele:

- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

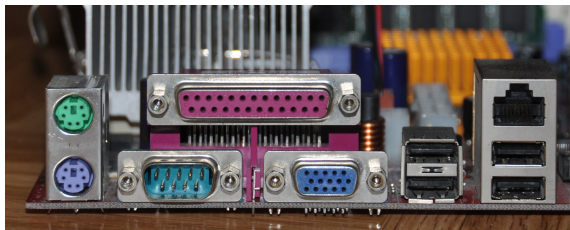


# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

Standard-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics



# 5 Bus-Systeme

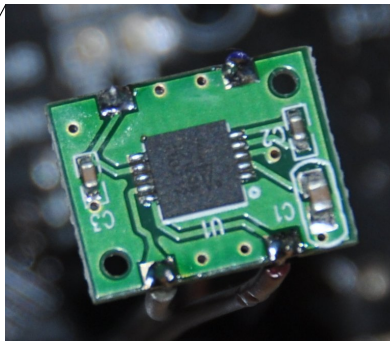
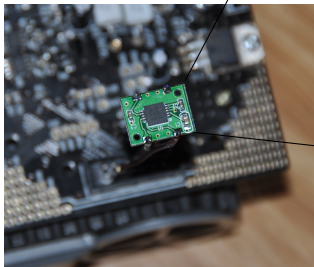
## 5.1 Was sind Bus-Systeme?

Standard-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics

Minimal-Hardware:

- RS-232
- I<sup>2</sup>C (TWI)
- SPI



# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

<i>seriell</i>	jedes Bit einzeln übertragen
<i>parallel</i>	mehrere Bits gleichzeitig
<i>synchron</i>	Abgleich über Steuerleitung: <i>Takt</i>
<i>asynchron</i>	Abgleich über Zeitvereinbarungen
<i>Punkt-zu-Punkt</i>	genau zwei Teilnehmer
<i>busfähig</i>	mehrere Teilnehmer, mit <i>Adressierung</i>

- I<sup>2</sup>C: seriell, synchron, mit Adressierung
- RS-232: seriell, asynchron, Punkt-zu-Punkt
- RS-485, USB, CAN: seriell, asynchron, mit Adressierung
- SPI: seriell, synchron, Punkt-zu-Punkt oder mit Adressierung

# 5 Bus-Systeme

## 5.2 RS-232

seriell

- *TX*: 1 Leitung für Daten
- *RX*: ggf. 1 Leitung für Daten in der anderen Richtung
- *GND*: gemeinsame *Masse*
- evtl. zusätzliche Steuerleitungen

asynchron

- *keine* Taktleitung für Abgleich, wann Daten anliegen
- Stattdessen: Abgleich über Zeitvereinbarungen

→ Jeder Teilnehmer braucht eine eigene Zeitbasis.

Punkt-zu-Punkt

- nur 2 Teilnehmer vorgesehen

## 5.2 RS-232

Synchronisation

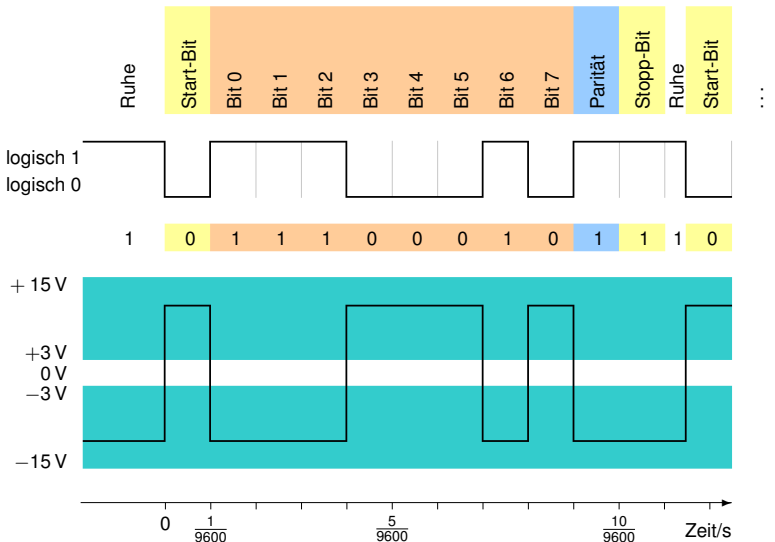
Daten

Check

9600 Baud, 8 Daten-Bits, ungerade Parität, 1 Stopp-Bit

Beispiel-Daten: ASCII „G“ = 71 = 0100 0111 binär

Übertragung der Daten von rechts (Bit 0) nach links (Bit 7)



# 5 Bus-Systeme

## 5.3 I<sup>2</sup>C (TWI)

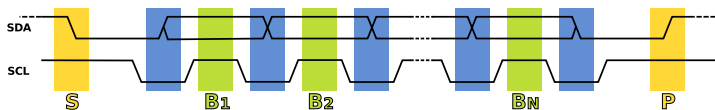
I<sup>2</sup>C = Inter-IC; TWI = Two-Wire-Interface

seriell

- **SDA**: 1 Leitung für Daten (in beiden Richtungen)
- **SCL**: Taktleitung (Clock)
- **GND**: gemeinsame Masse
- evtl. **VCC**: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung



busfähig

- **Master** initiiert Kommunikation und steuert Taktleitung
- erstes gesendetes Byte: **Adresse** des Teilnehmers
- 2 Adressen pro Teilnehmer: Lesen/Schreiben

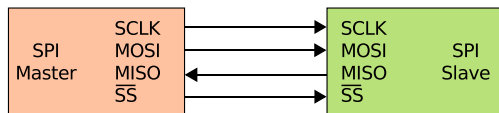
# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät



synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt

# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

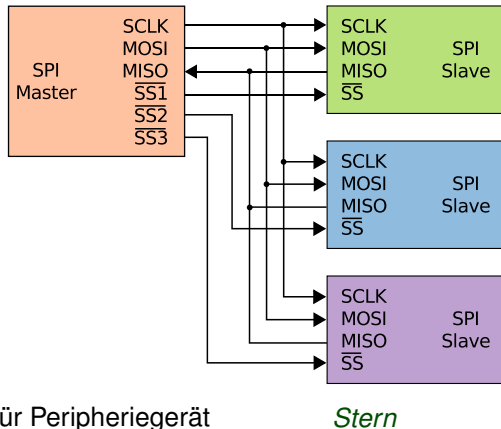
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt





# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

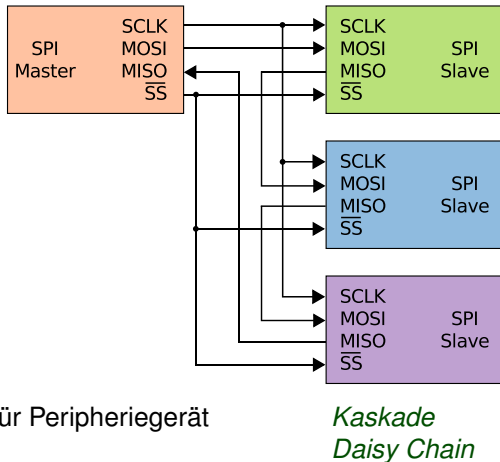
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

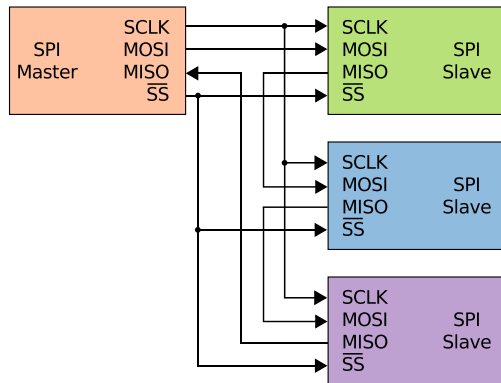
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



*Kaskade  
Daisy Chain*

Slave gibt MOSI-Input um 1 Takt verzögert an MISO aus → Master setzt „im richtigen Moment“  $\overline{SS}$

# 5 Bus-Systeme

## 5.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

- Steuerung von Motoren
- Nutzung als allgemeines Protokoll zur Übertragung analoger Werte

# 5 Bus-Systeme

## 5.6 Sonstiges

### Matrix-Schaltung

- möglichst viele Aktoren/Sensoren  
über möglichst wenige digital Inputs abfragen  
bzw. über möglichst wenige digitale Outputs steuern
- Beispiele: LED-Felder, Tastaturen

### R/2R-Netzwerk

- möglichst viele digitale Inputs  
über einen einzigen analogen Input abfragen
- Beispiele: Tastaturen

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
  - 5.1 Was sind Bus-Systeme?
  - 5.2 RS-232
  - 5.3 I<sup>2</sup>C (TWI)
  - 5.4 SPI
  - 5.5 PWM
  - 5.6 Sonstiges
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?
  - 6.2 Echtzeitprogrammierung
  - 6.3 Multitasking
  - 6.4 Ressourcen
  - 6.5 Prioritäten

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.



## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

→ „Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*





## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

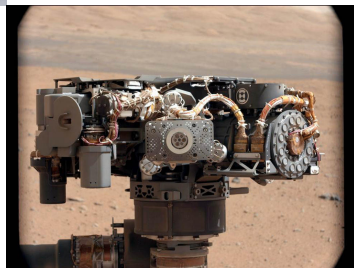
- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“  
→ *keine Echtzeit*

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit  
→ Na und?

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

**„Verschwendung von Rechenzeit – na und?“**



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

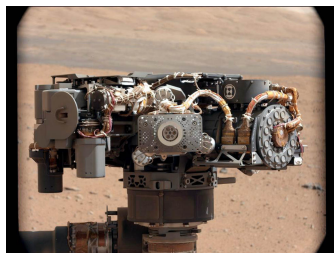
### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren

→ **Echtzeitprogrammierung**

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware

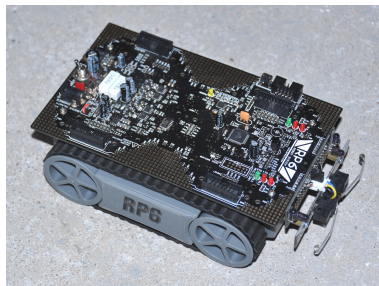


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software





## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

12. Januar 2021

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?
  - 6.2 Echtzeitprogrammierung
  - 6.3 Multitasking
  - 6.4 Ressourcen
  - 6.5 Prioritäten

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

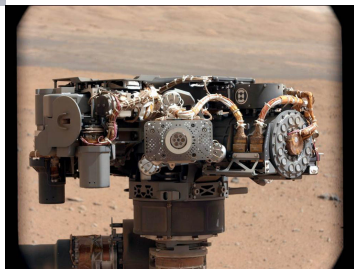
- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

→ „Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*





## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“  
→ *keine Echtzeit*

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit  
→ Na und?

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

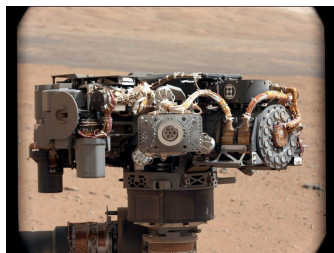
### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren

→ **Echtzeitprogrammierung**

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware



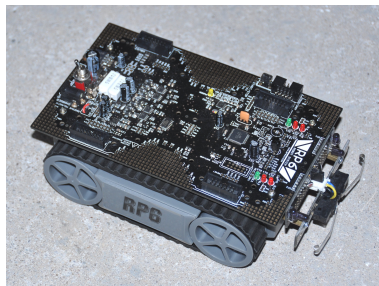


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: Benutzerprogramm

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: 1 Benutzerprogramm

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)



## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse
- Ausblick: Zuteilung von Rechenzeit = wichtiger Spezialfall  
allgemein: Zuteilung von Ressourcen

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

## 6 Echtzeit

### 6.3 Multitasking

Qualitätsaspekte beim Multitasking

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:

*Latenz* vs. *Jitter*

vs. *Durchsatz*

- *Latenz*: interaktive Anwendungen
- *Jitter*: Echtzeitanwendungen
- *Durchsatz*: Stapelverarbeitung

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe*

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)



# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits  
(IPv4) bzw. Traffic-Class-Bits (IPv6)  
im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel  
griechisch: *sema* – Zeichen, *pherein* – tragen  
„Eisenbahnsignal“

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\rightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann  
  
englisch: *mutual exclusion* – wechselseitiger Ausschluß  
spezieller binärer Semaphor: nur „Besitzer“ darf freigeben

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\longrightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock* (*busy waiting*)  
leichtgewichtige Alternative zu Kontextwechsel  
englisch: *spin* – rotieren, *lock* Sperre  
*busy waiting* auf etwas Schnelles, z. B. auf einen Semaphor  
Hardware-Unterstützung: Prüfen, ob Variable bestimmten Wert hat;  
wenn ja, auf anderen Wert setzen; andere Prozessoren solange anhalten

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein



## 6 Echtzeit

### 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

# 6 Echtzeit

## 6.4 Ressourcen

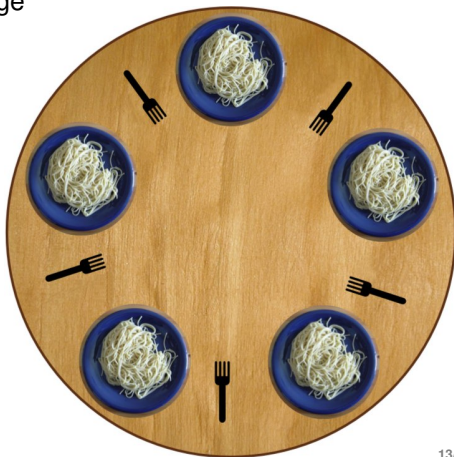
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**



# 6 Echtzeit

## 6.4 Ressourcen

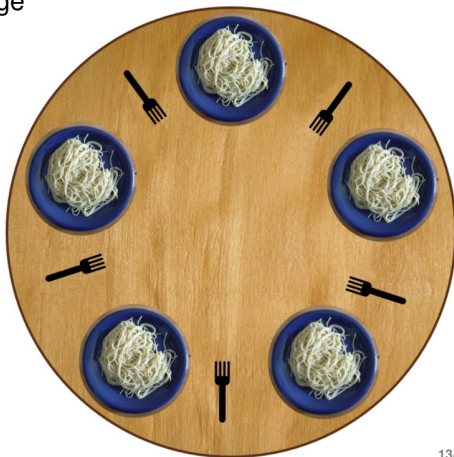
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge (z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**  
schweigen → **Deadlock**  
philosophieren weiter → **Livelock**



## 6 Echtzeit

### 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- Exklusivität
- *hold and wait*
- Entzug nicht möglich
- zirkuläre Blockade

# 6 Echtzeit

## 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- |                        |   |
|------------------------|---|
| • Exklusivität         | → Spooling  |
| • <i>hold and wait</i> | → simultane Zuteilung                             |
| • Entzug nicht möglich | → Prozesse suspendieren, beenden, <i>Rollback</i> |
| • zirkuläre Blockade   | → Reihenfolge abhängig von Ressourcen             |

# 6 Echtzeit

## 6.5 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert; Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Prozeß mit positivem Zähler gibt, bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

→ *dynamische Prioritätenvergabe*:  
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

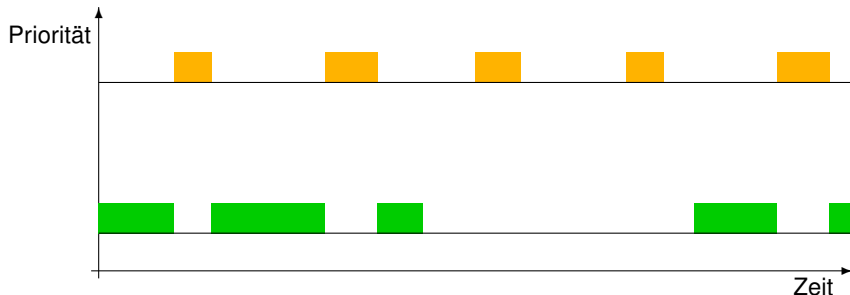
→ *statische Prioritätenvergabe*

# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

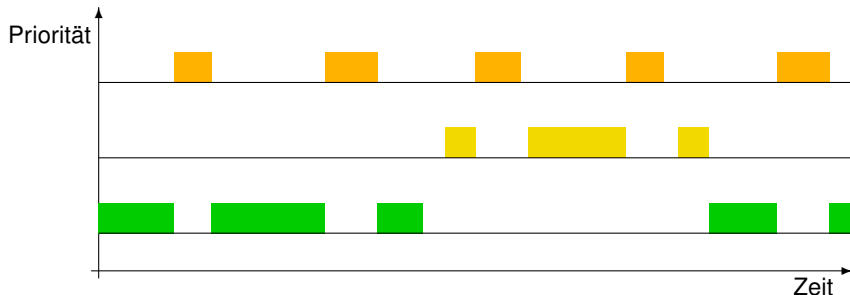


# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

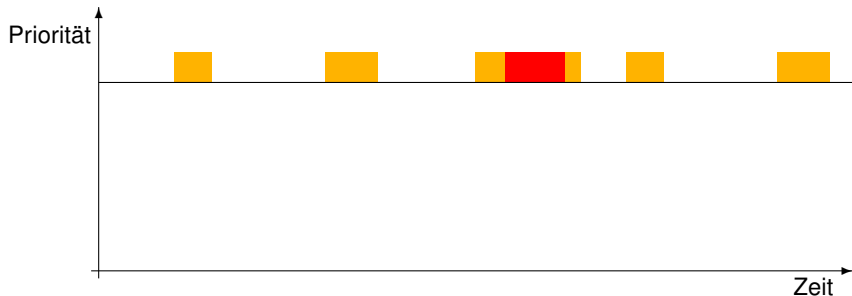
- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.





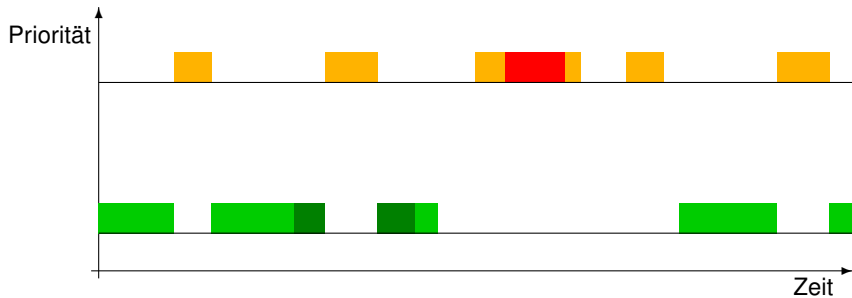
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



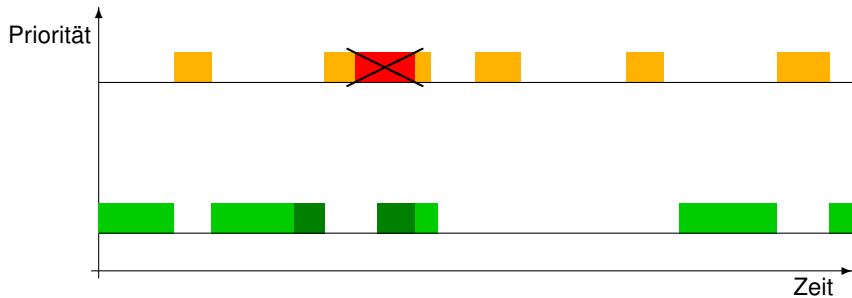
## 6 Echtzeit

## 6.5 Prioritäten und Ressourcen



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



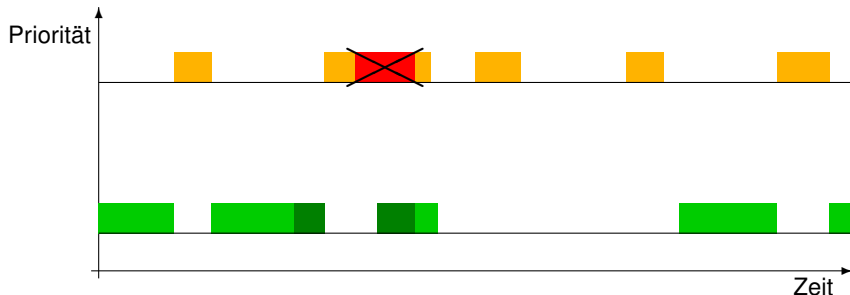
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



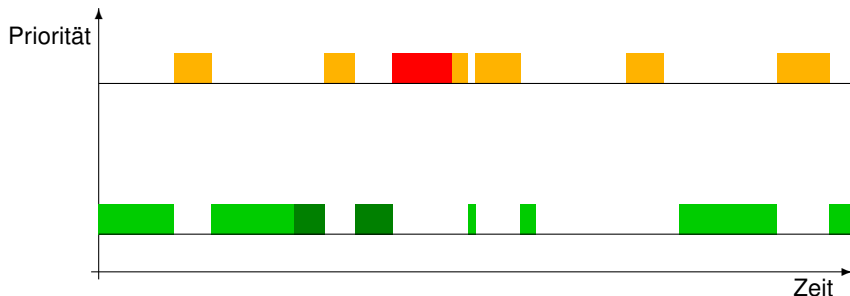
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



## 6 Echtzeit

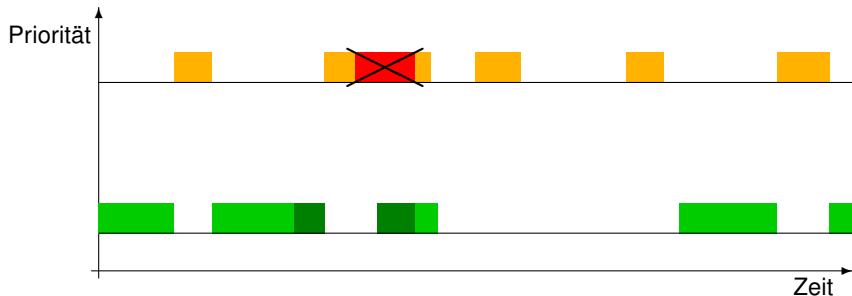
### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

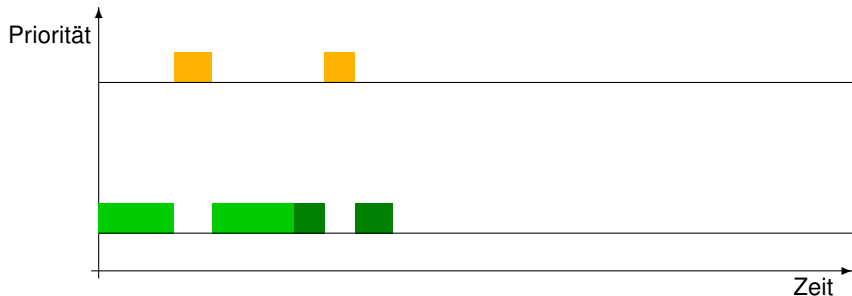
*unbegrenzte Prioritätsinversion*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*



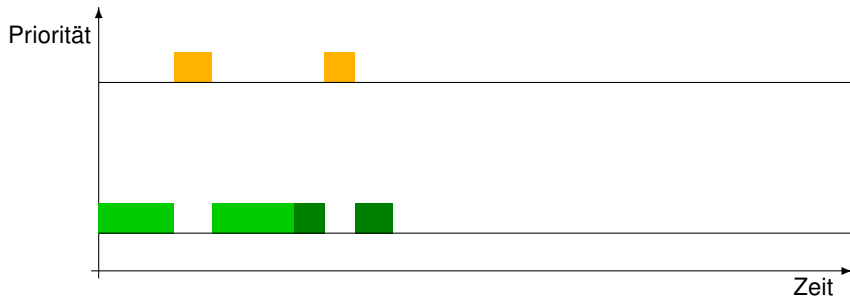


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

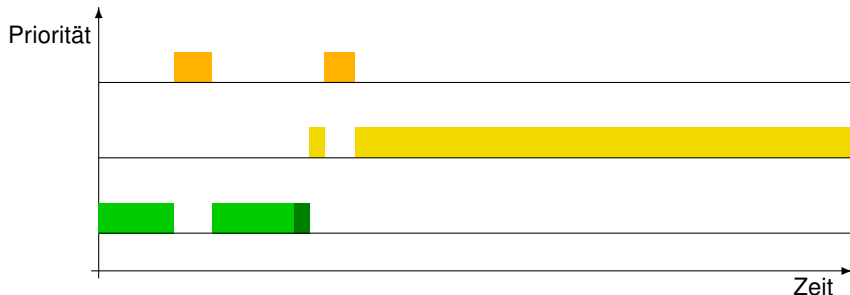


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



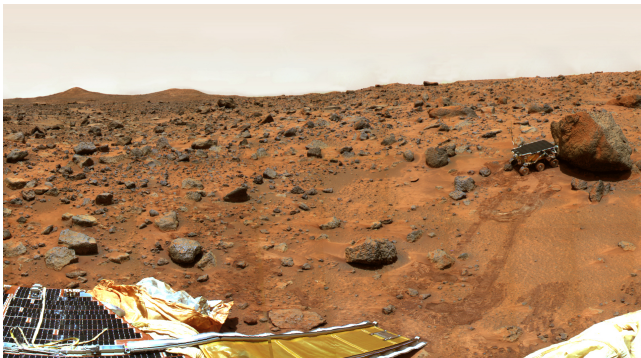
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

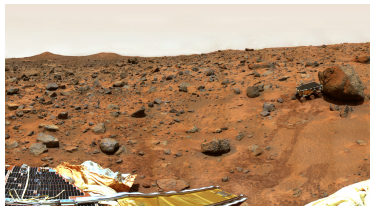
—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.

[http://research.microsoft.com/en-us/um/people/mbj/Mars\\_Pathfinder/](http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/)



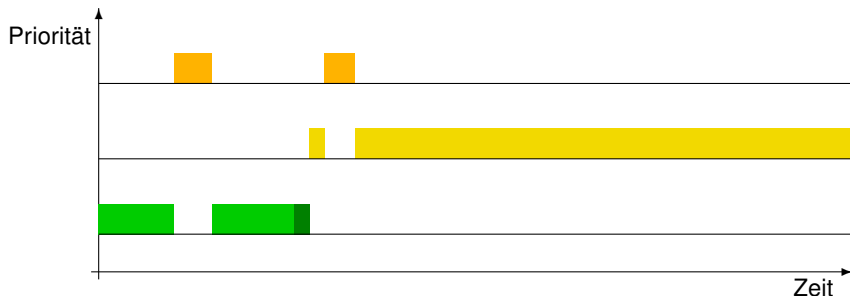
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



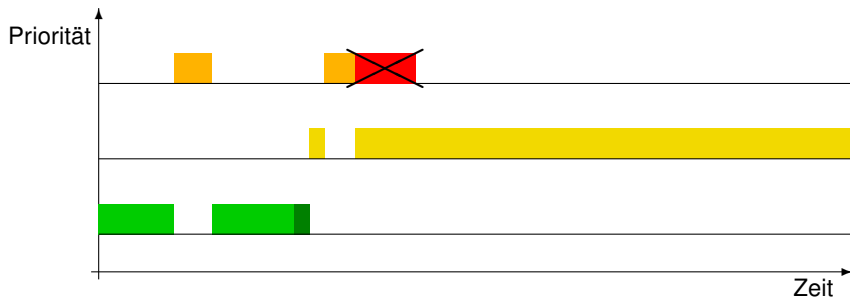
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*





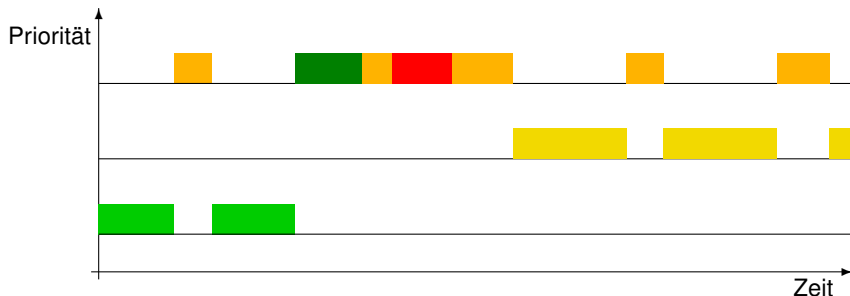
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Ceiling – Prioritätsobergrenze*





## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
- } nur möglich, wenn  
Mutexe im Spiel sind

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
  - *Priority Aging*  
Die Priorität wächst mit der Wartezeit.
- } nur möglich, wenn  
Mutexe im Spiel sind

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

19. Januar 2021

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?**
  - 6.2 Echtzeitprogrammierung**
  - 6.3 Multitasking**
  - 6.4 Ressourcen**
  - 6.5 Prioritäten**

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse
- Ausblick: Zuteilung von Rechenzeit = wichtiger Spezialfall  
allgemein: Zuteilung von Ressourcen

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

## Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Tasks wird dekrementiert; Task mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Task mit positivem Zähler gibt, bekommen alle Tasks gemäß ihrer Priorität neue Zähler zugewiesen.
- *keine* harte Echtzeit



# Zombies

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? —> Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.
- Aber? → Wenn sich Zombie-Prozesse anhäufen, deutet dies auf einen Prozeß hin, der andere Prozesse erzeugt und anschließend „hängt“.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.
- Aber? → Wenn sich Zombie-Prozesse anhäufen, deutet dies auf einen Prozeß hin, der andere Prozesse erzeugt und anschließend „hängt“.
- Beispiel: Datenträger entfernt, zugreifender Prozeß „hängt“.  
→ Tochterprozesse werden zu Zombies.

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
  - *Latenz*: interaktive Anwendungen
  - *Jitter*: Echtzeitanwendungen
  - *Durchsatz*: Stapelverarbeitung

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits (IPv4) bzw. Traffic-Class-Bits (IPv6) im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*



# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits  
(IPv4) bzw. Traffic-Class-Bits (IPv6)  
im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel  
griechisch: *sema* – Zeichen, *pherein* – tragen  
„Eisenbahnsignal“

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\longrightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann  
  
englisch: *mutual exclusion* – wechselseitiger Ausschluß  
spezieller binärer Semaphor: nur „Besitzer“ darf freigeben

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\rightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel  
englisch: *spin* – rotieren, *lock* Sperre  
*busy waiting* auf etwas Schnelles, z. B. auf einen Semaphor  
Hardware-Unterstützung: Prüfen, ob Variable bestimmten Wert hat;  
wenn ja, auf anderen Wert setzen; andere Prozessoren solange anhalten

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren – Beispiel: `linux-3.7rc1`

- *Semaphor*  
kernel/semaphor.c  
drivers/usb/core/file.c
- *Mutex*  
kernel/mutex.c  
drivers/usb/serial/usb-serial.c
- *Spinlock*  
kernel/spinlock.c  
kernel/semaphor.c, kernel/mutex.c

**Beispiel:** `usb_serial_get_by_index()` – serielle Schnittstelle reservieren  
Datei `linux-3.7-rc1/drivers/usb/serial/usb-serial.c`, ab Zeile 62

```
struct usb_serial *usb_serial_get_by_index (unsigned index)
{
    struct usb_serial *serial;
    mutex_lock (&table_lock); ← exklusiven Zugriff auf Tabelle sichern
    serial = serial_table[index];
    if (serial)
    {
        mutex_lock (&serial->disc_mutex);
        if (serial->disconnected)
        {
            mutex_unlock (&serial->disc_mutex);
            serial = NULL;
        }
        else
            kref_get (&serial->kref);
    }
    mutex_unlock (&table_lock); ← exklusiven Zugriff auf Tabelle wieder freigeben
    return serial;
}
```

mutex\_lock() – Ressourcen beimproben, nichtleeren  
 Datei linux-3.7-rc1/drivers/tty/ttyld/usb-serial.c, ab Zeile 62

```
void __sched mutex_lock (struct mutex *lock)
{
    might_sleep ();
    __mutex_lock_path (lock->lock->count, __mutex_lock_sleeppath);
    mutex_set_owner (lock);
}
```

Datei linux-3.7-rc1/include/linux/mutex.h, ab Zeile 54  
 Macro-Definition für \_\_mutex\_lock\_path (expandiert)

```
Assembler:
lock-dec (lock->count)
jns 1
call __mutex_lock_sleeppath
t:
```

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 266

```
static __used noinline void __sched
__mutex_lock_sleeppath (atomic_t *lock_count)
{
    struct mutex *lock = container_of (lock_count, struct mutex, count);
    __mutex_lock_common (lock, TASK_UNINTERRUPTIBLE, 0,
        NULL, _RET_IP_);
}
```

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 132

```
static inline int __sched
__mutex_lock_common (struct mutex *lock, long state, unsigned int subclass,
    struct task_struct *task, struct task_struct *waiter, unsigned long ip)
{
    struct task_struct *task = current;
    struct mutex *waiter = waiter;
    preempt_disable ();
```

```
    mutex_acquire_wait (lock->dep_map, subclass, 0, wait_lock_ip);
```

```
    <-- <--
```

```
spin_lock_mutex (lock->wait_lock, flag);
```

← exklusiven Zugriff auf Mutex sichern

```
debug_mutex_lock_common (lock, &waiter);
debug_mutex_add_waiter (lock, &waiter, task, thread_info (task));
```

```
/* add waiting task to the end of the waitqueue (FFDC) */
let, add_wait (&waiter, lock, &lock->wait_list);
waiter->task = task;
```

```
if (atomic_xchg (lock->count, -1) == 1)
    goto done;
```

```
lock_contended (lock->dep_map, ip);
```

```
for (;;)
{
    <--
```

```
/* Later try to take the lock again - this is needed even if
 * we get here for the first time (shortly after failing to
 * acquire the lock), to make sure that we get a wakeup once
 * it's unlocked. Later on, if we sleep, this is the
 * operation that gives us the lock. We sleep if it is -1, so
 * that when we release the lock, we properly wake up the
 * other waiters.
 */
```

```
if (atomic_xchg (lock->count, -1) == 1)
    break;
```

```
/* got a signal? (This code gets eliminated in the
 * TASK_UNINTERRUPTIBLE case.)
 */
```

```
if (unlikely (signal_pending_state (state, task))
```

```
{
    mutex_remove_waiter (lock, &waiter, task, thread_info (task));
    mutex_release (lock->dep_map, 1, ip);
    spin_unlock_mutex (lock->wait_lock, flag);
```

```
    debug_mutex_free_waiter (&waiter);
    preempt_enable ();
    return -EINTR;
}
```

```
__wait_task_state (task, state);
```

```
/* didn't get the lock, go to sleep: */
```

```
spin_unlock_mutex (lock->wait_lock, flag);
schedule_preempt_disabled ();
spin_lock_mutex (lock->wait_lock, flag);
```

```
done:
```

```
lock_acquired (lock->dep_map, ip);
/* got the lock - register */
mutex_remove_waiter (lock, &waiter, current_thread_info ());
mutex_set_owner (lock);
```

```
/* set it to 0 if there are no waiters left: */
```

```
if (likely (!task) (lock->wait_list))
    atomic_set (lock->count, 0);
```

```
spin_unlock_mutex (lock->wait_lock, flag);
```

← exklusiven Zugriff auf Mutex wieder freigeben

```
debug_mutex_free_waiter (&waiter);
preempt_enable ();
```

```
return 0;
```



spin\_lock\_mutex() – Mutex beanspruchen, notfalls busy waiting  
Detail linux-3.7-rc1/kernel/mutex.h, ab Zeile 12

```
#define spin_lock_mutex(lock, flag) {\n    do {\n        spin_lock(lock); \n        (void) (flag); \n    } while (0)\n}
```

Detail linux-3.7-rc1/kernel/spinlock.h, ab Zeile 283

```
static inline void spin_lock (spinlock_t *lock)\n{\n    raw_spin_lock (&lock->lock);\n}
```

Detail linux-3.7-rc1/kernel/spinlock.h, Zeile 170

```
#define raw_spin_lock(lock) _raw_spin_lock(lock)
```

Detail linux-3.7-rc1/include/linux/spinlock\_api\_smp.h, Zeile 47

```
#define _raw_spin_lock(lock) __raw_spin_lock(lock)
```

Detail linux-3.7-rc1/kernel/spinlock.c, ab Zeile 46 (excerpt):

```
void __lockfunc __raw_spin_lock (spinlock_t *lock)\n{\n    for (;;) {\n        preempt_disable ();\n        if (likely (do_raw_spin_trylock (lock)))\n            break;\n        preempt_enable ();\n\n        if (!lock) --> break; lock\n        (lock) --> break; lock = 1;\n        while (raw_spin_trylock (lock) && lock) --> break; lock\n        arch_spin_relax (&lock->raw_lock);\n    }\n    (lock) --> break; lock = 0;\n}
```

Detail linux-3.7-rc1/include/linux/spinlock.h, ab Zeile 150:

```
static inline int do_raw_spin_trylock (raw_spinlock_t *lock)\n{\n    return arch_spin_trylock (&lock->raw_lock);\n}
```

Detail arch/i686/include/asm/spinlock.h, ab Zeile 116:

```
static __always_inline int arch_spin_trylock (arch_spinlock_t *lock)\n{\n    return __ticket_spin_trylock (lock);\n}
```

Detail arch/i686/include/asm/spinlock.h, ab Zeile 65:

```
static __always_inline int __ticket_spin_trylock (arch_spinlock_t *lock)\n{\n    arch_spinlock_t old, new;\n\n    old.ticket = ACCESS_ONCE (lock->ticket);\n    if (old.ticket <= 1 - old.ticket) {\n        return 0;\n\n        new.head_tail = old.head_tail + (1 << TICKET_SHIFT);\n        // cmpxchg is a full barrier, so nothing can move before it v/\n        return cmpxchg (&lock->head_tail, old.head_tail, new.head_tail) == old.head;\n    }\n}
```

Detail arch/i686/include/asm/compichg.h, ab Zeile 147:

```
#define cmpchg(ptr, old, new) {\n    __cmpchg (ptr, old, new, sizeof (<ptr>))\n}
```

Detail arch/i686/include/asm/compichg.h, ab Zeile 131:

```
#define __cmpchg(ptr, old, new, size) {\n    __raw_cmpchg ((ptr), (old), (new), (size), LOCK_PREFIX)\n}
```

Detail arch/i686/include/asm/compichg.h, ab Zeile 110:

```
asm volatile (lock "cmpxchgl %2,%1"\n    : "+l" ("ptr"), "=l" ("new")\n    : "r" ("new"), "r" ("old")\n    : "memory");
```

atomarer und exklusiver  
Zugriff auf Spinlock  
durch Hardware-Unterstützung

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

# 6 Echtzeit

## 6.4 Ressourcen

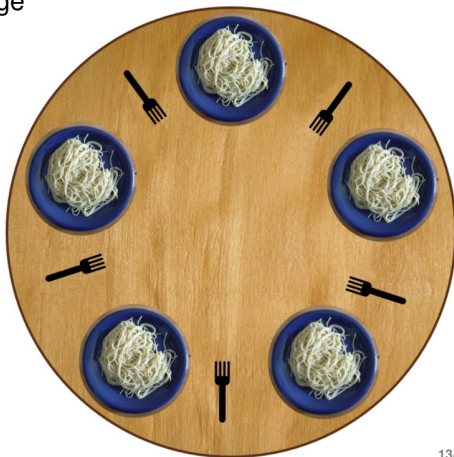
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**  
schweigen → **Deadlock**  
philosophieren weiter → **Livelock**



# 6 Echtzeit

## 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- Exklusivität → Spooling
- *hold and wait* → simultane Zuteilung
- Entzug nicht möglich → Prozesse suspendieren, beenden, *Rollback*
- zirkuläre Blockade → Reihenfolge abhängig von Ressourcen

# 6 Echtzeit

## 6.5 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert; Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Prozeß mit positivem Zähler gibt, bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

→ *dynamische Prioritätenvergabe*:  
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

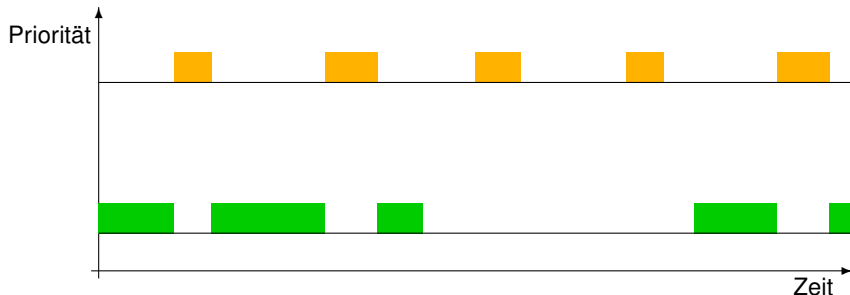
→ *statische Prioritätenvergabe*

# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

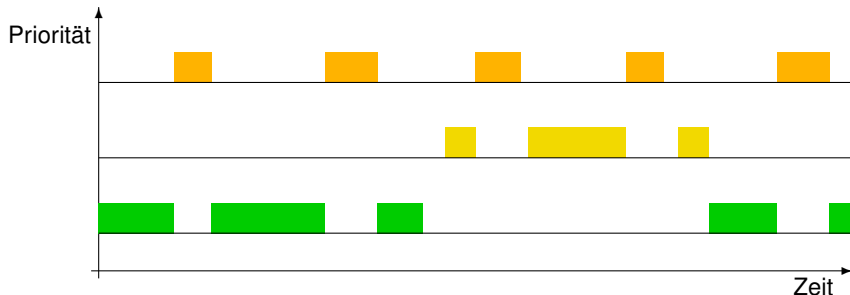


# 6 Echtzeit

## 6.5 Prioritäten

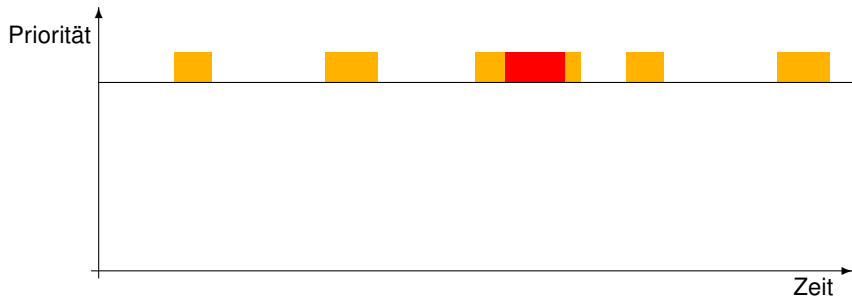
### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.



## 6 Echtzeit

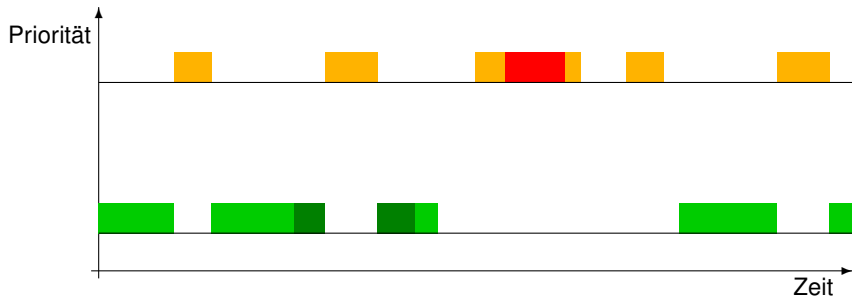
### 6.5 Prioritäten und Ressourcen





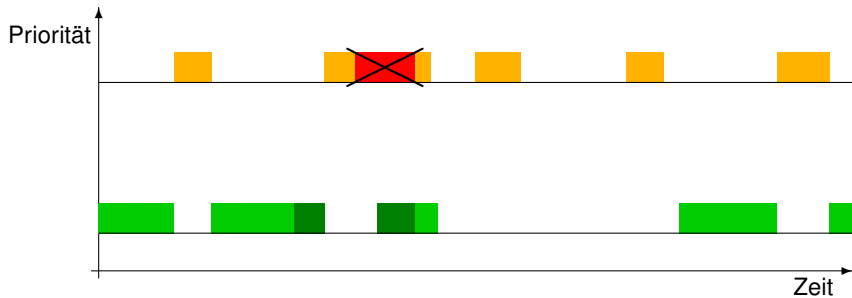
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



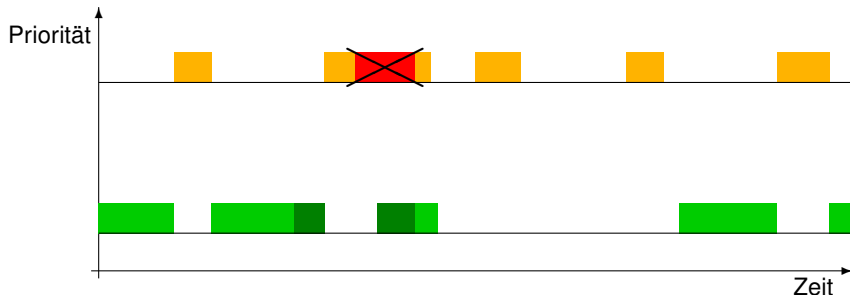
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



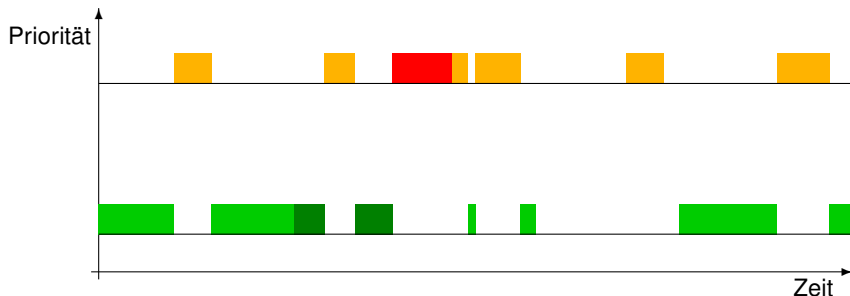
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



## 6 Echtzeit

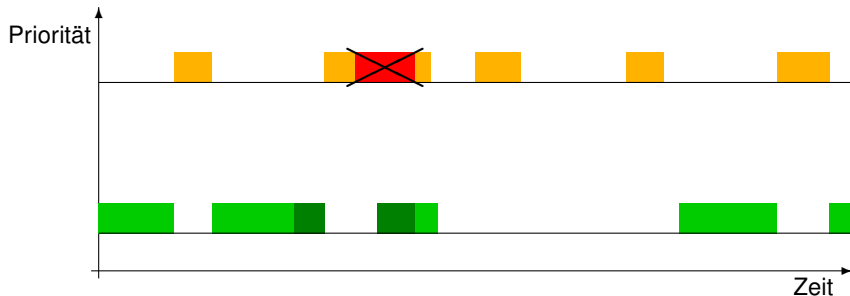
### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

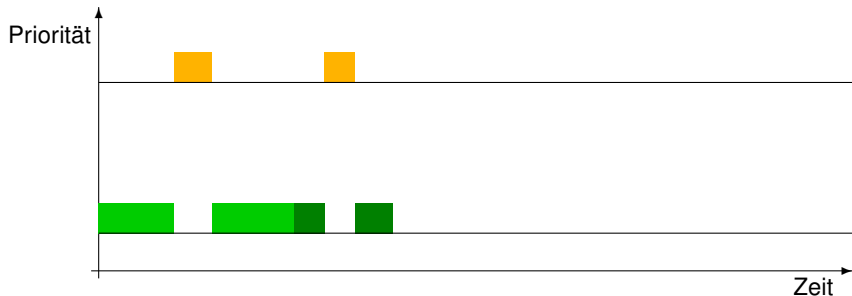
*unbegrenzte Prioritätsinversion*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

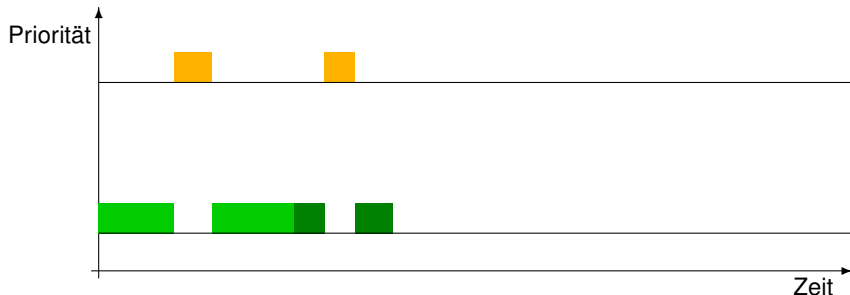


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



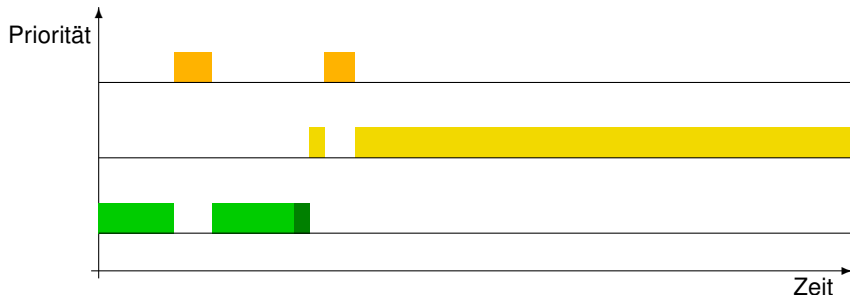


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



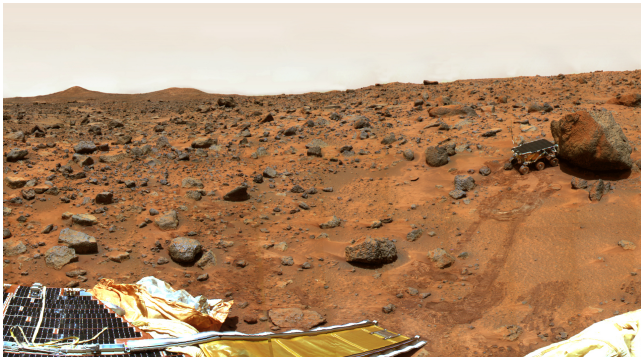
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

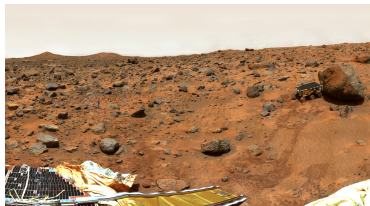
—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.

[http://research.microsoft.com/en-us/um/people/mbj/Mars\\_Pathfinder/](http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/)



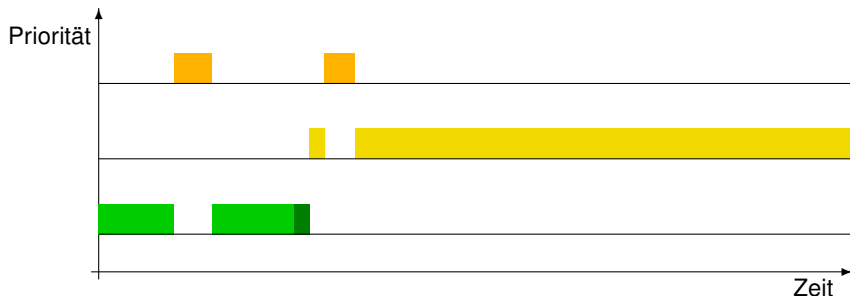
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



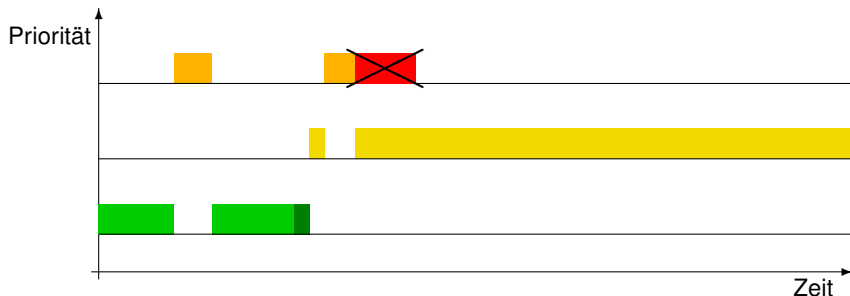
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



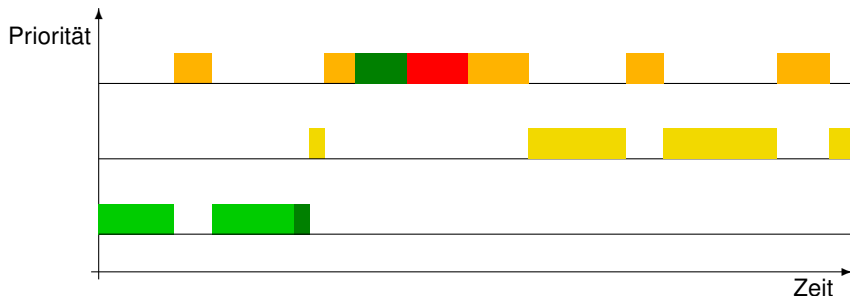
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



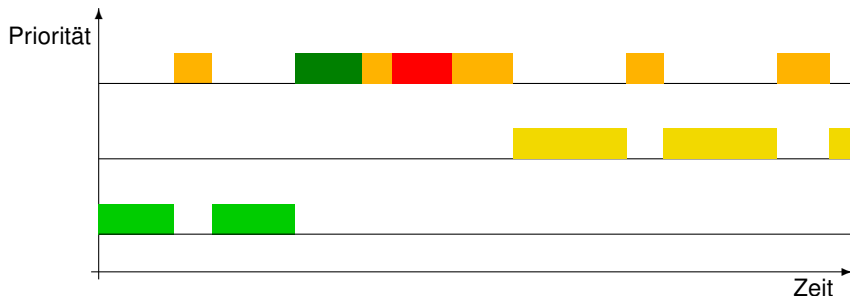
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Ceiling – Prioritätsobergrenze*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
- } nur möglich, wenn  
Mutexe im Spiel sind

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
  - *Priority Aging*  
Die Priorität wächst mit der Wartezeit.
- } nur möglich, wenn  
Mutexe im Spiel sind

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

26. Januar 2021

# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?**
  - 6.2 Echtzeitprogrammierung**
  - 6.3 Multitasking**
  - 6.4 Ressourcen**
  - 6.5 Prioritäten**
  - 6.6 Threads**

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse
- **Ausblick:** Zuteilung von Rechenzeit = wichtiger Spezialfall  
allgemein: Zuteilung von Ressourcen

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

## Linux 0.01

- Timer-Interrupt:  
Task mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Task mit positivem Zähler gibt, bekommen alle Tasks gemäß ihrer Priorität neue Zähler zugewiesen.
- *keine* harte Echtzeit

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

## Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Tasks wird dekrementiert; Task mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen lafbereiten Task mit positivem Zähler gibt, bekommen alle Tasks gemäß ihrer Priorität neue Zähler zugewiesen.
- *keine* harte Echtzeit

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.
- Aber? → Wenn sich Zombie-Prozesse anhäufen, deutet dies auf einen Prozeß hin, der andere Prozesse erzeugt und anschließend „hängt“.
- Beispiel: Datenträger entfernt, zugreifender Prozeß „hängt“.  
→ Tochterprozesse werden zu Zombies.



# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
  - *Latenz*: interaktive Anwendungen
  - *Jitter*: Echtzeitanwendungen
  - *Durchsatz*: Stapelverarbeitung

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits (IPv4) bzw. Traffic-Class-Bits (IPv6) im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits  
(IPv4) bzw. Traffic-Class-Bits (IPv6)  
im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel  
griechisch: *sema* – Zeichen, *pherein* – tragen  
„Eisenbahnsignal“

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\longrightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann  
  
englisch: *mutual exclusion* – wechselseitiger Ausschluß  
spezieller binärer Semaphor: nur „Besitzer“ darf freigeben

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\longrightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock* (*busy waiting*)  
leichtgewichtige Alternative zu Kontextwechsel  
englisch: *spin* – rotieren, *lock* Sperre  
*busy waiting* auf etwas Schnelles, z. B. auf einen Semaphor  
Hardware-Unterstützung: Prüfen, ob Variable bestimmten Wert hat;  
wenn ja, auf anderen Wert setzen; andere Prozessoren solange anhalten

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren – Beispiel: `linux-3.7rc1`

- *Semaphor*  
kernel/semaphor.c  
drivers/usb/core/file.c
- *Mutex*  
kernel/mutex.c  
drivers/usb/serial/usb-serial.c
- *Spinlock*  
kernel/spinlock.c  
kernel/semaphor.c, kernel/mutex.c

**Beispiel:** `usb_serial_get_by_index()` – serielle Schnittstelle reservieren  
Datei `linux-3.7-rc1/drivers/usb/serial/usb-serial.c`, ab Zeile 62

```
struct usb_serial *usb_serial_get_by_index (unsigned index)
{
    struct usb_serial *serial;
    mutex_lock (&table_lock); ← exklusiven Zugriff auf Tabelle sichern
    serial = serial_table[index];
    if (serial)
    {
        mutex_lock (&serial->disc_mutex);
        if (serial->disconnected)
        {
            mutex_unlock (&serial->disc_mutex);
            serial = NULL;
        }
        else
            kref_get (&serial->kref);
    }
    mutex_unlock (&table_lock); ← exklusiven Zugriff auf Tabelle wieder freigeben
    return serial;
}
```



mutex\_lock() – Ressourcen beimprimieren, nichtle werden  
 Datei linux-3.7-rc1/drivers/tty/ttyld/usb-serial.c, ab Zeile 62

```
void __sched mutex_lock (struct mutex *lock)
{
    might_sleep ();
    __mutex_lock_path (lock->lock->count, __mutex_lock_sleeppath);
    mutex_set_owner (lock);
}
```

Datei linux-3.7-rc1/include/linux/mutex.h, ab Zeile 54  
 Macro-Definition für \_\_mutex\_lock\_path (expandiert)

```
Assembler:
lock-dec (lock->count)
jns 1
call __mutex_lock_sleeppath
t:
```

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 266

```
static __used noinline void __sched
__mutex_lock_sleeppath (atomic_t *lock_count)
{
    struct mutex *lock = container_of (lock_count, struct mutex, count);
    __mutex_lock_common (lock, TASK_UNINTERRUPTIBLE, 0,
        NULL, _RET_IP_);
}
```

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 132

```
static inline int __sched
__mutex_lock_common (struct mutex *lock, long state, unsigned int subclass,
    struct task_struct *task, struct task_struct *waiter, unsigned long ip)
{
    struct task_struct *task = current;
    struct mutex *waiter = waiter;
    unsigned long flags;

    preempt_disable ();
    mutex_acquire_post (lock->dep_map, subclass, 0, next_lock, ip);

    <-- <-- <--

    spin_lock_mutex (lock->wait_lock, flags); // exklusiven Zugriff auf Mutex sichern
    debug_mutex_lock_common (lock, &waiter);
    debug_mutex_add_waiter (lock, &waiter, task, thread_info (task));

    <-- add waiting task to the end of the waitqueue (FFQC) <--
    list_add_tail (&waiter, list, &lock->wait_list);
    waiter->task = task;

    if (atomic_xchg (lock->count, -1) == 1)
        goto done;

    lock_contended (lock->dep_map, ip);

    for (;;)
    {
        <--
        <-- Later try to take the lock again -- this is needed even if
        <-- we get here for the first time (shortly after failing to
        <-- acquire the lock), to make sure that we get a wakeup once
        <-- it's unlocked. Later on, if we sleep, this is the
        <-- operation that gives us the lock. We sleep if it is -1, so
        <-- that when we release the lock, we properly wake up the
        <-- other waiters.
        <--
        if (atomic_xchg (lock->count, -1) == 1)
            break;

        <--
        <-- got a signal? (This code gets eliminated in the
        <-- TASK_UNINTERRUPTIBLE case.)
        <--
        if (unlikely (signal_pending_state (state, task)))
        {
            mutex_remove_waiter (lock, &waiter, task, thread_info (task));
            mutex_release (lock->dep_map, 1, ip);
            spin_unlock_mutex (lock->wait_lock, flags);

            debug_mutex_free_waiter (&waiter);
            preempt_enable ();
            return -EINTR;
        }
        __set_task_state (task, state);

        <-- didn't get the lock, go to sleep: <--
        spin_unlock_mutex (lock->wait_lock, flags);
        schedule_preempt_disabled ();
        spin_lock_mutex (lock->wait_lock, flags);
    }

done:
    lock_acquired (lock->dep_map, ip);
    <-- got the lock -- signal <--
    mutex_remove_waiter (lock, &waiter, current, thread_info ());
    mutex_set_owner (lock);

    <-- set it to 0 if there are no waiters left: <--
    if (likely (!list_empty (lock->wait_list)))
        atomic_set (lock->count, 0);

    spin_unlock_mutex (lock->wait_lock, flags);
    debug_mutex_free_waiter (&waiter);
    preempt_enable ();

    return 0;
}
```

relativen Zugriff auf Mutex  
 unter freigegeben

spin\_lock\_mutex() – Mutex beanspruchen, notfalls busy waiting  
Detail linux-3.7-rc1/kernel/mutex.h, ab Zeile 12

```
#define spin_lock_mutex(lock, flag) {\n    do {\n        spin_lock(lock); \n        (void) (flag); \n    } while (0)\n}
```

Detail linux-3.7-rc1/kernel/spinlock.h, ab Zeile 283

```
static inline void spin_lock (spinlock_t *lock)\n{\n    raw_spin_lock (&lock->lock);\n}
```

Detail linux-3.7-rc1/kernel/spinlock.h, Zeile 170

```
#define raw_spin_lock(lock) _raw_spin_lock(lock)
```

Detail linux-3.7-rc1/include/linux/spinlock\_api\_smp.h, Zeile 47

```
#define _raw_spin_lock(lock) __raw_spin_lock(lock)
```

Detail linux-3.7-rc1/kernel/spinlock.c, ab Zeile 46 (expendiert):

```
void __lockfunc __raw_spin_lock (spinlock_t *lock)\n{\n    for (;;) {\n        preempt_disable ();\n        if (likely (do_raw_spin_trylock (lock)))\n            break;\n        preempt_enable ();\n\n        if (!lock) --> break; lock\n        (lock) --> break; lock = 1;\n        while (raw_spin_trylock (lock) && lock) --> break; lock\n        arch_spin_relax (&lock->raw_lock);\n    }\n    (lock) --> break; lock = 0;\n}
```

Detail linux-3.7-rc1/include/linux/spinlock.h, ab Zeile 150:

```
static inline int do_raw_spin_trylock (raw_spinlock_t *lock)\n{\n    return arch_spin_trylock (&lock->raw_lock);\n}
```

Detail arch/i686/include/asm/spinlock.h, ab Zeile 116:

```
static __always_inline int arch_spin_trylock (arch_spinlock_t *lock)\n{\n    return __ticket_spin_trylock (lock);\n}
```

Detail arch/i686/include/asm/spinlock.h, ab Zeile 65:

```
static __always_inline int __ticket_spin_trylock (arch_spinlock_t *lock)\n{\n    arch_spinlock_t old, new;\n\n    old.ticket = ACCESS_ONCE (lock->ticket);\n    if (old.ticket <= 1 - old.ticket) {\n        return 0;\n\n        new.head_tail = old.head_tail + (1 << TICKET_SHIFT);\n        // cmpxchg is a full barrier, so nothing can move before it v/\n        return cmpxchg (&lock->head_tail, old.head_tail, new.head_tail) == old.head;\n    }\n}
```

Detail arch/i686/include/asm/cmpxchg.h, ab Zeile 147:

```
#define cmpxchg(ptr, old, new) {\n    __cmpxchg (ptr, old, new, sizeof (*ptr))\n}
```

Detail arch/i686/include/asm/cmpxchg.h, ab Zeile 131:

```
#define __cmpxchg(ptr, old, new, size)\n    __raw_cmpxchg ((ptr), (old), (new), (size), LOCK_PREFIX)
```

Detail arch/i686/include/asm/cmpxchg.h, ab Zeile 110:

```
asm volatile (lock "cmpxchgl %2,%1"\n    : "+l" (%_new), "+l" (%_old)\n    : "r" (%_new), "r" (%_old)\n    : "memory");
```

atomarer und exklusiver  
Zugriff auf Spinlock  
durch Hardware-Unterstützung

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

# 6 Echtzeit

## 6.4 Ressourcen

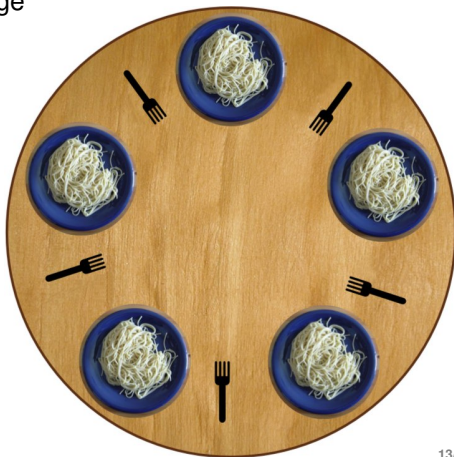
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge (z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**  
schweigen → **Deadlock**  
philosophieren weiter → **Livelock**



# 6 Echtzeit

## 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- Exklusivität → Spooling
- *hold and wait* → simultane Zuteilung
- Entzug nicht möglich → Prozesse suspendieren, beenden, *Rollback*
- zirkuläre Blockade → Reihenfolge abhängig von Ressourcen

# 6 Echtzeit

## 6.5 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert; Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Prozeß mit positivem Zähler gibt, bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

→ *dynamische Prioritätenvergabe*:  
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

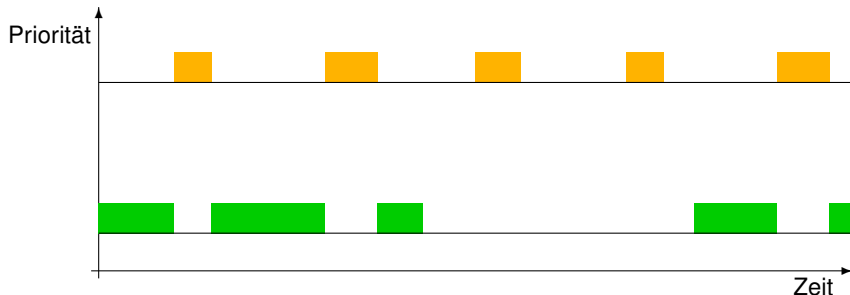
→ *statische Prioritätenvergabe*

# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

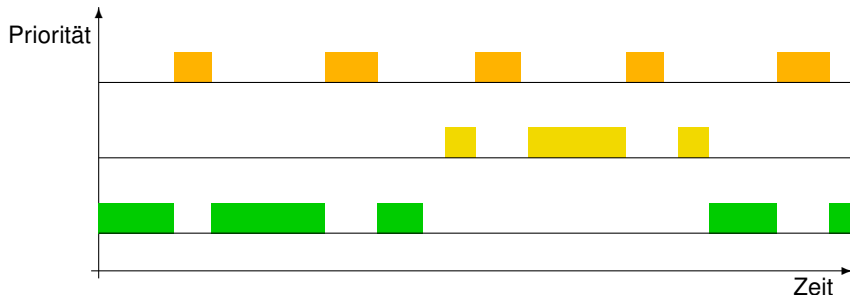


# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

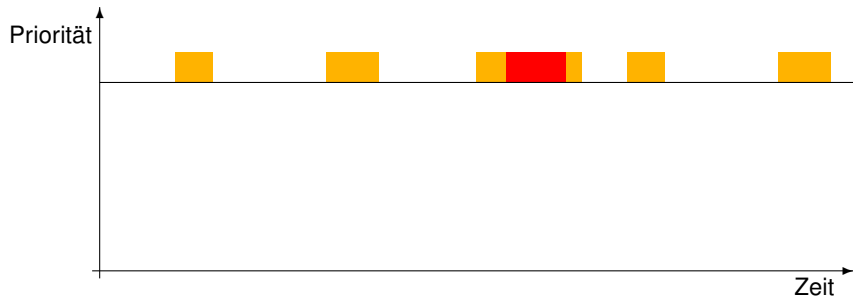
- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.





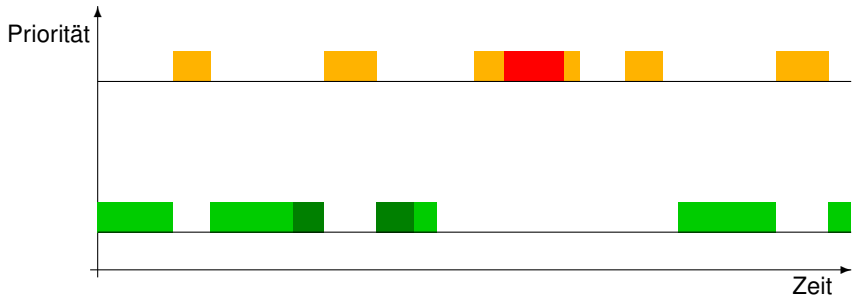
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



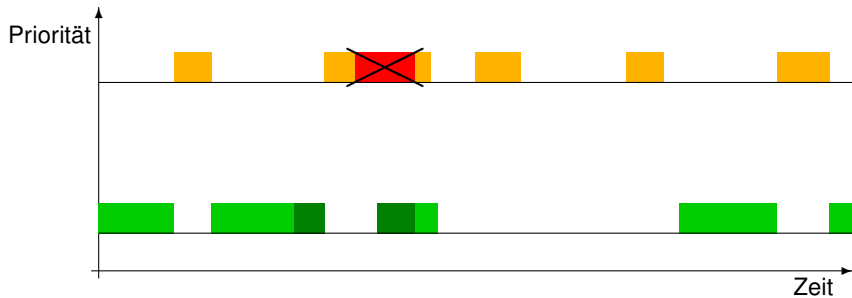
## 6 Echtzeit

## 6.5 Prioritäten und Ressourcen



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



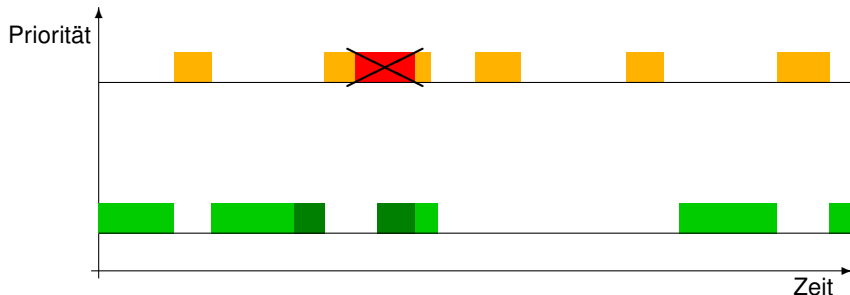
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



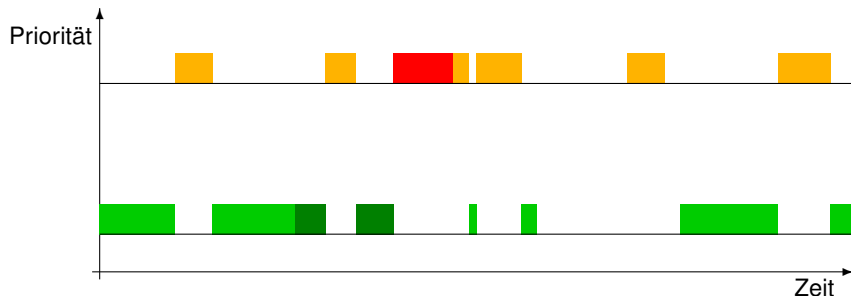
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



## 6 Echtzeit

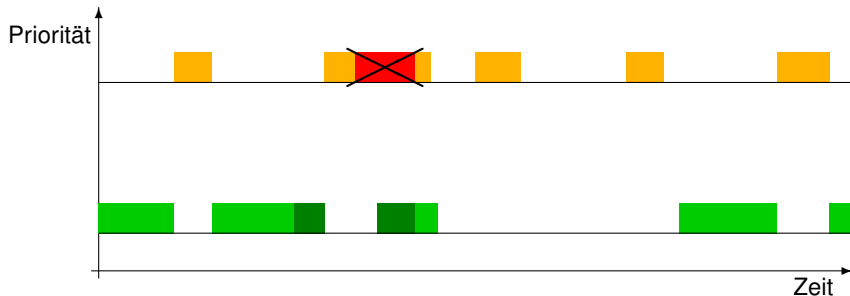
### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

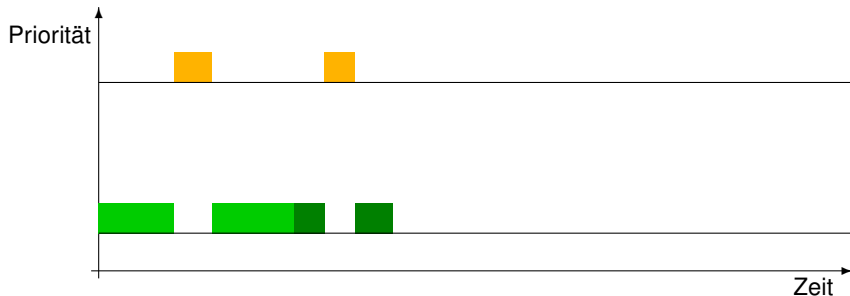
*unbegrenzte Prioritätsinversion*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*



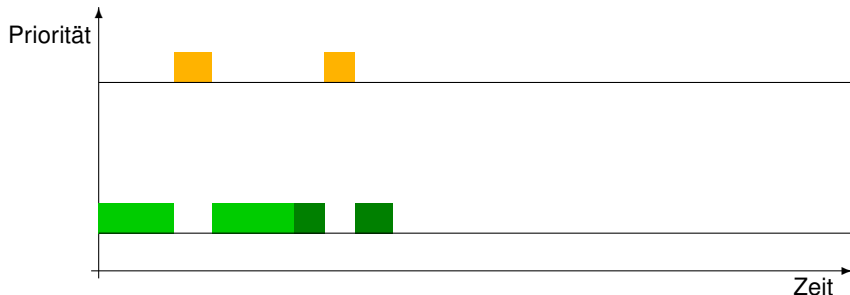


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

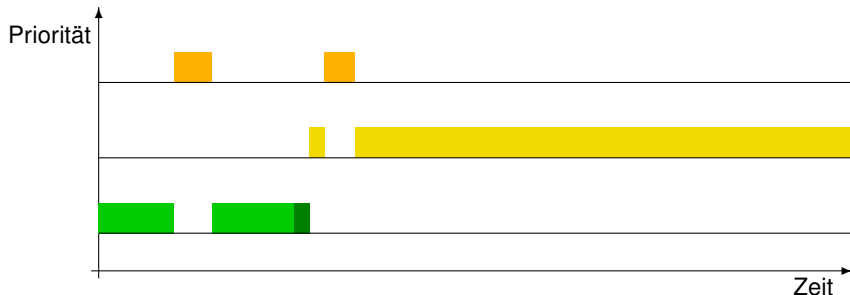


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



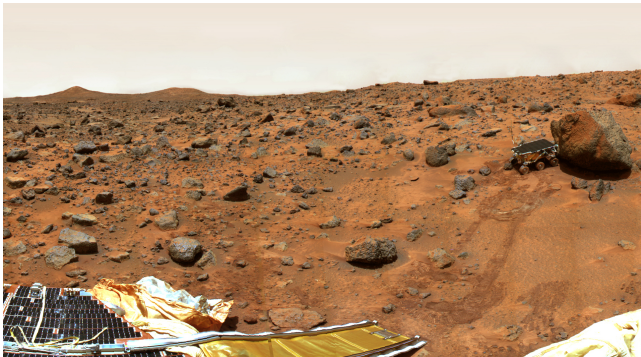
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

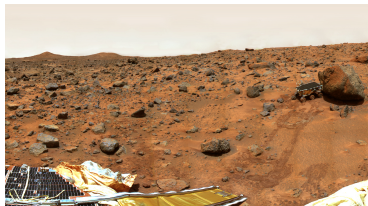
—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.

[http://research.microsoft.com/en-us/um/people/mbj/Mars\\_Pathfinder/](http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/)



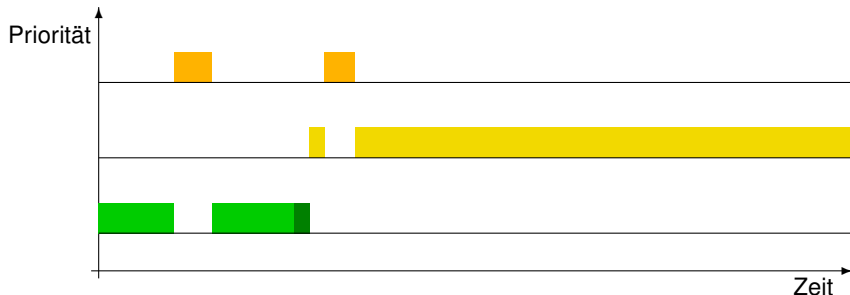
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



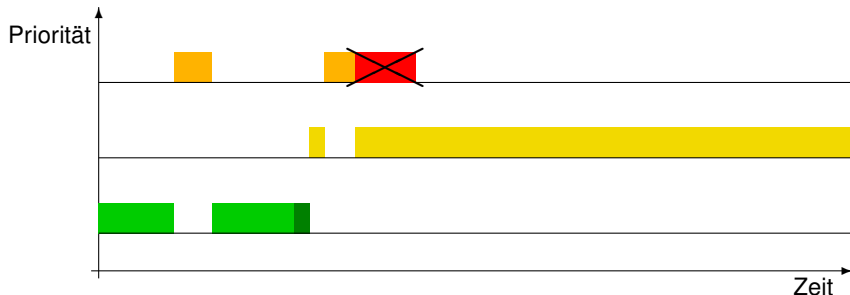
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



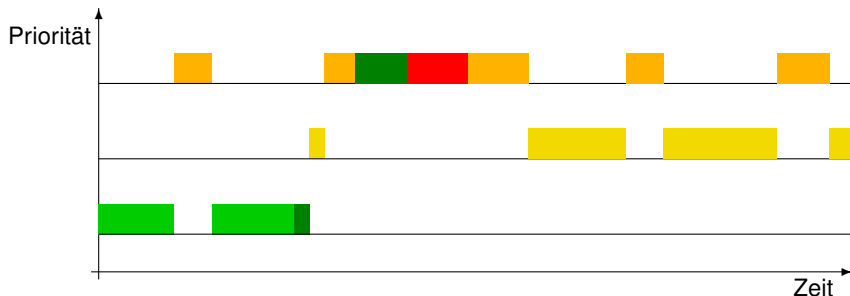
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



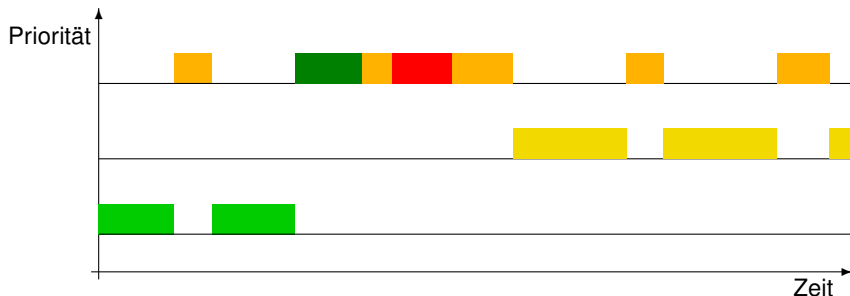
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Ceiling – Prioritätsobergrenze*





## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

} nur möglich, wenn  
Mutexe im Spiel sind

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
  - *Priority Aging*  
Die Priorität wächst mit der Wartezeit.
- } nur möglich, wenn  
Mutexe im Spiel sind

# 6 Echtzeit

## 6.6 Threads

### Parallelisierung

- Programme parallel ausführen: Prozesse  
jeweils eigener Speicherbereich
- Programmteile parallel ausführen: Threads  
gemeinsamer Speicherbereich
- in beiden Fällen:  
Synchronisation über Semaphoren, insbesondere Mutexe

### Ziele

- einfachere Programmierung nebenläufiger Aktionen  
→ neue Probleme durch Synchronisation
- Nutzung mehrerer Prozessoren/Kerne

# 6 Echtzeit

## 6.6 Threads

### Anwendungen

- GUI-Programmierung
- Kommunikations-Software
- Numerik
  
- Beispiel: Mumble-Partybot
- Übungsaufgabe: Chat-Software
- Übungsaufgabe: responsive GUI

# 6 Echtzeit

## 6.6 Threads

### Übungsaufgabe: Chat-Software

- Benutzer tippen Chat-Messages ein.
- Nach Eingabe von ENTER soll die Message als ganze „animiert“ ausgegeben werden, d. h. mit einer geringen Verzögerung zwischen den einzelnen Zeichen, z.B. 1/20 Sekunde.
- Ein Bot schreibt gelegentlich die aktuelle Uhrzeit in den Chat.
- Irgendwann wird der Bot mit der Textausgabe kollidieren. Das Programm soll dieses Problem einerseits demonstrieren und andererseits zeigen, wie es sich durch Verwendung von Mutexen beheben läßt.
- Vorgabe: Lösung in C.  
Optional: Lösung in weiteren Programmiersprachen.
- Hinweis: <https://www.thegeekstuff.com/2012/05/c-mutex-examples/>

## 6 Echtzeit

### 6.6 Threads

Übungsaufgabe: responsive GUI

- Ein Haupt-Thread wartet auf Ereignisse.
- Wenn diese eintreten, bearbeitet es sie und initiiert einen Neuaufbau des Bildschirms.
- Der Bildschirmaufbau erfolgt in separaten Threads, damit der Haupt-Thread sofort weiterarbeiten kann.
- Wenn durch die weitere Verarbeitung der Bildschirmaufbau hinfällig geworden ist, sollte der Haupt-Thread das dem Bildschirmaufbau-Thread signalisieren, damit dieser abbrechen kann.

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

2. Februar 2021



# Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
- 4 Versionsverwaltungssysteme**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1** Was ist Echtzeit?
  - 6.2** Echtzeitprogrammierung
  - 6.3** Multitasking
  - 6.4** Ressourcen
  - 6.5** Prioritäten
  - 6.6** Threads
- 7 Verschlüsselung**
  - 7.1** Verschlüsselungsverfahren
  - 7.2** Zertifizierung von Schlüsseln

# 6 Echtzeit

## 6.5 Threads

### Parallelisierung

- Programme parallel ausführen: Prozesse  
jeweils eigener Speicherbereich
- Programmteile parallel ausführen: Threads  
gemeinsamer Speicherbereich
- in beiden Fällen:  
Synchronisation über Semaphoren, insbesondere Mutexe

### Ziele

- einfachere Programmierung nebenläufiger Aktionen  
→ neue Probleme durch Synchronisation
- Nutzung mehrerer Prozessoren/Kerne

# 6 Echtzeit

## 6.5 Threads

### Anwendungen

- GUI-Programmierung
- Kommunikations-Software
- Numerik
- Beispiel: Mumble-Partybot
- Übungsaufgabe: Chat-Software
- Übungsaufgabe: responsive GUI

# 6 Echtzeit

## 6.5 Threads

### Übungsaufgabe: Chat-Software

- Benutzer tippen Chat-Messages ein.
- Nach Eingabe von ENTER soll die Message als ganze „animiert“ ausgegeben werden, d. h. mit einer geringen Verzögerung zwischen den einzelnen Zeichen, z.B. 1/20 Sekunde.
- Ein Bot schreibt gelegentlich die aktuelle Uhrzeit in den Chat.
- Irgendwann wird der Bot mit der Textausgabe kollidieren. Das Programm soll dieses Problem einerseits demonstrieren und andererseits zeigen, wie es sich durch Verwendung von Mutexen beheben läßt.
- Vorgabe: Lösung in C.  
Optional: Lösung in weiteren Programmiersprachen.
- Hinweis: <https://www.thegeekstuff.com/2012/05/c-mutex-examples/>

## 6 Echtzeit

### 6.5 Threads

Übungsaufgabe: responsive GUI

- Ein Haupt-Thread wartet auf Ereignisse.
- Wenn diese eintreten, bearbeitet es sie und initiiert einen Neuaufbau des Bildschirms.
- Der Bildschirmaufbau erfolgt in separaten Threads, damit der Haupt-Thread sofort weiterarbeiten kann.
- Wenn durch die weitere Verarbeitung der Bildschirmaufbau hinfällig geworden ist, sollte der Haupt-Thread das dem Bildschirmaufbau-Thread signalisieren, damit dieser abbrechen kann.

## 7 Verschlüsselung

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1).x$  Motivation

Man kann auch mit sehr merkwürdigen Objekten wie mit „ganz normalen“ Zahlen rechnen.



## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1).x$  Motivation

Man kann auch mit sehr merkwürdigen Objekten wie mit „ganz normalen“ Zahlen rechnen.

Anwendungen:

- Funktionsweise von Computern ( $\longrightarrow$  Rechnertechnik)
- Fehlererkennung
- Fehlerkorrektur
- Verschlüsselung
- Digitale Signaturen

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1) \cdot (x + 1)$  Gruppen

**Definition:** Sei  $G$  eine Menge,  $*$  eine Verknüpfung auf  $G$ . Wenn

- $\forall a, b, c \in G: (a * b) * c = a * (b * c)$  (Assoziativgesetz),
- $\exists e \in G: \forall a \in G: a * e = e * a = a$  (neutrales Element),
- $\forall a \in G: \exists a^{-1} \in G: a * a^{-1} = a^{-1} * a = e$  (inverses Element),

dann heißt  $(G, *)$  eine *Gruppe*.

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1) \cdot (x + 1)$  Gruppen

**Definition:** Sei  $G$  eine Menge,  $*$  eine Verknüpfung auf  $G$ . Wenn

- $\forall a, b, c \in G: (a * b) * c = a * (b * c)$  (Assoziativgesetz),
- $\exists e \in G: \forall a \in G: a * e = e * a = a$  (neutrales Element),
- $\forall a \in G: \exists a^{-1} \in G: a * a^{-1} = a^{-1} * a = e$  (inverses Element),

dann heißt  $(G, *)$  eine *Gruppe*.

**Definition:** Sei  $(G, *)$  eine Gruppe. Wenn zusätzlich

- $\forall a, b \in G: a * b = b * a$  (Kommutativgesetz),

dann heißt  $(G, *)$  eine *kommutative Gruppe*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1) \cdot (x + 2)$ Ringe

**Definition:** Sei  $R$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $R$ . Wenn

- $(R, +)$  eine kommutative Gruppe ist,
- $\forall a, b, c \in R: (a \cdot b) \cdot c = a \cdot (b \cdot c)$  (Assoziativgesetz),
- $\forall a, b, c \in R: (a + b) \cdot c = a \cdot c + b \cdot c$  und  $a \cdot (b + c) = a \cdot b + a \cdot c$  (Distributivgesetze),

dann heit  $(R, +, \cdot)$  ein *Ring*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1) \cdot (x + 2)$ Ringe

**Definition:** Sei  $R$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $R$ . Wenn

- $(R, +)$  eine kommutative Gruppe ist,
- $\forall a, b, c \in R: (a \cdot b) \cdot c = a \cdot (b \cdot c)$  (Assoziativgesetz),
- $\forall a, b, c \in R: (a + b) \cdot c = a \cdot c + b \cdot c$  und  $a \cdot (b + c) = a \cdot b + a \cdot c$  (Distributivgesetze),

dann heit  $(R, +, \cdot)$  ein *Ring*.

**Definition:** Sei  $(R, +, \cdot)$  ein Ring. Wenn zustzlich

- $\forall a, b \in R: a \cdot b = b \cdot a$  (Kommutativgesetz),

dann heit  $(R, +, \cdot)$  ein *kommutativer Ring*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1) \cdot (x + 2)$ Ringe

**Definition:** Sei  $R$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $R$ . Wenn

- $(R, +)$  eine kommutative Gruppe ist,
- $\forall a, b, c \in R: (a \cdot b) \cdot c = a \cdot (b \cdot c)$  (Assoziativgesetz),
- $\forall a, b, c \in R: (a + b) \cdot c = a \cdot c + b \cdot c$  und  $a \cdot (b + c) = a \cdot b + a \cdot c$  (Distributivgesetze),

dann heit  $(R, +, \cdot)$  ein *Ring*.

**Definition:** Sei  $(R, +, \cdot)$  ein Ring. Wenn zustzlich

- $\forall a, b \in R: a \cdot b = b \cdot a$  (Kommutativgesetz),

dann heit  $(R, +, \cdot)$  ein *kommutativer Ring*.

**Definition:** Sei  $(R, +, \cdot)$  ein (kommutativer) Ring. Wenn zustzlich

- ein  $e \in R$  existiert, so da fr alle  $a \in R$  gilt:  $a \cdot e = e \cdot a = a$  (neutrales Element),

dann heit  $(R, +, \cdot)$  ein *(kommutativer) Ring mit 1*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1).(x + 3)$ Körper

**Definition:** Sei  $K$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $K$ . Wenn

- $(K, +, \cdot)$  ein Ring mit 1 ist und
- $(K \setminus \{0\}, \cdot)$  eine kommutative Gruppe ist,

dann heißt  $(K, +, \cdot)$  ein *Körper*.

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall



# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:  
Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:

Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...

unsicher

Rijndael = AES, RC4 = CipherSaber

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:  
Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...  
unsicher  
Rijndael = AES, RC4 = CipherSaber

Problem: geheimer Kanal für Schlüsselaustausch erforderlich

Lösung: *asymmetrische Verschlüsselung*

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

Problem: Verfahren sind langsam

Lösung: *hybride Verschlüsselung*:

asymmetrisches Verfahren verschlüsselt symmetrischen *Sitzungsschlüssel*

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

Problem: Verfahren sind langsam

Lösung: *hybride Verschlüsselung*:

asymmetrisches Verfahren verschlüsselt symmetrischen *Sitzungsschlüssel*

Problem: nicht-manipulierbarer Kanal für Schlüsselaustausch erforderlich

Lösung: *Zertifizierung*

# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust

# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust – kann auch hierarchische Baumstruktur sein

OpenPGP: E-Mail, spezielle Anwendungen, . . .

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: persönliche Entscheidung



# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust – kann auch hierarchische Baumstruktur sein

OpenPGP: E-Mail, spezielle Anwendungen, ...

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: persönliche Entscheidung

S/MIME: Webseiten, E-Mail, spezielle Anwendungen, ...

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: wird vom Anbieter vorgegeben

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

9. Februar 2021

# Eingebettete Systeme

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

- 1 Einführung
- 2 Einführung in Unix
- 3 TCP/IP in der Praxis
- 4 Versionsverwaltungssysteme
- 5 Bus-Systeme
- 6 Echtzeit
- 7 Verschlüsselung
  - 7.( $x^2 - 1$ ) Der Herr der Ringe
  - 7.1 Verschlüsselungsverfahren
  - 7.2 Zertifizierung von Schlüsseln

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1).x$  Motivation

Man kann auch mit sehr merkwürdigen Objekten wie mit „ganz normalen“ Zahlen rechnen.

Anwendungen:

- Funktionsweise von Computern ( $\longrightarrow$  Rechnertechnik)
- Fehlererkennung
- Fehlerkorrektur
- Verschlüsselung
- Digitale Signaturen

## 7 Verschlüsselung

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1) \cdot (x + 1)$  Gruppen

**Definition:** Sei  $G$  eine Menge,  $*$  eine Verknüpfung auf  $G$ . Wenn

- $\forall a, b, c \in G: (a * b) * c = a * (b * c)$  (Assoziativgesetz),
- $\exists e \in G: \forall a \in G: a * e = e * a = a$  (neutrales Element),
- $\forall a \in G: \exists a^{-1} \in G: a * a^{-1} = a^{-1} * a = e$  (inverses Element),

dann heißt  $(G, *)$  eine *Gruppe*.

**Definition:** Sei  $(G, *)$  eine Gruppe. Wenn zusätzlich

- $\forall a, b \in G: a * b = b * a$  (Kommutativgesetz),

dann heißt  $(G, *)$  eine *kommutative Gruppe*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1) \cdot (x + 2)$ Ringe

**Definition:** Sei  $R$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $R$ . Wenn

- $(R, +)$  eine kommutative Gruppe ist,
- $\forall a, b, c \in R: (a \cdot b) \cdot c = a \cdot (b \cdot c)$  (Assoziativgesetz),
- $\forall a, b, c \in R: (a + b) \cdot c = a \cdot c + b \cdot c$  und  $a \cdot (b + c) = a \cdot b + a \cdot c$  (Distributivgesetze),

dann heißt  $(R, +, \cdot)$  ein *Ring*.

**Definition:** Sei  $(R, +, \cdot)$  ein Ring. Wenn zusätzlich

- $\forall a, b \in R: a \cdot b = b \cdot a$  (Kommutativgesetz),

dann heißt  $(R, +, \cdot)$  ein *kommutativer Ring*.

## 7. $(x^2 - 1)$ Der Herr der Ringe: Manchmal ist $1 + 1 = 0$ .

### 7. $(x^2 - 1) \cdot (x + 2)$ Ringe

**Definition:** Sei  $R$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $R$ . Wenn

- $(R, +)$  eine kommutative Gruppe ist,
- $\forall a, b, c \in R: (a \cdot b) \cdot c = a \cdot (b \cdot c)$  (Assoziativgesetz),
- $\forall a, b, c \in R: (a + b) \cdot c = a \cdot c + b \cdot c$  und  $a \cdot (b + c) = a \cdot b + a \cdot c$  (Distributivgesetze),

dann heißt  $(R, +, \cdot)$  ein *Ring*.

**Definition:** Sei  $(R, +, \cdot)$  ein Ring. Wenn zusätzlich

- $\forall a, b \in R: a \cdot b = b \cdot a$  (Kommutativgesetz),

dann heißt  $(R, +, \cdot)$  ein *kommutativer Ring*.

Sei  $(R, +, \cdot)$  ein (kommutativer) Ring. Wenn zusätzlich

- ein  $e \in R$  existiert, so daß für alle  $a \in R$  gilt:  $a \cdot e = e \cdot a = a$  (neutrales Element),

dann heißt  $(R, +, \cdot)$  ein *(kommutativer) Ring mit 1*.

7.  $(x^2 - 1)$  Der Herr der Ringe: Manchmal ist  $1 + 1 = 0$ .

7.  $(x^2 - 1) \cdot (x + 3)$  Körper

**Definition:** Sei  $K$  eine Menge; seien  $+$  und  $\cdot$  Verknüpfungen auf  $K$ . Wenn

- $(K, +, \cdot)$  ein Ring mit 1 ist und
- $(K \setminus \{0\}, \cdot)$  eine kommutative Gruppe ist,

dann heißt  $(K, +, \cdot)$  ein *Körper*.



# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:  
Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:

Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...

unsicher

Rijndael = AES, RC4 = CipherSaber

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Symmetrische Verschlüsselung:*

Derselbe Schlüssel zum Ver- und Entschlüsseln

- Cäsar-Chiffre: monoalphabetische Substitution
- Vigenère-Chiffre: polyalphabetische Substitution
- Kryptanalyse: Kasiski-Test, Friedman-Test
- One-Time-Pad
- Pseudozufall
- spezieller Pseudozufall:  
Enigma, RC4, DES, 3DES, IDEA, Rijndael, Blowfish, Twofish, CAST5, ...  
unsicher  
Rijndael = AES, RC4 = CipherSaber

Problem: geheimer Kanal für Schlüsselaustausch erforderlich

Lösung: *asymmetrische Verschlüsselung*

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

Problem: Verfahren sind langsam

Lösung: *hybride Verschlüsselung*:

asymmetrisches Verfahren verschlüsselt symmetrischen *Sitzungsschlüssel*

# 7 Verschlüsselung

## 7.1 Verschlüsselungsverfahren

### *Asymmetrische Verschlüsselung:*

Verschiedene Schlüssel zum Ver- und Entschlüsseln

- mathematische Operation „schwierig“ umkehrbar
- Messung von „schwierig“: Landau-Symbol
- Beispiele:
  - Primfaktorzerlegung schwieriger als Multiplikation von Primzahlen
  - Logarithmus schwieriger als Potenz
- Verfahren:
  - RSA, DSA, ElGamal, ECRSA, ...

Problem: Verfahren sind langsam

Lösung: *hybride Verschlüsselung*:

asymmetrisches Verfahren verschlüsselt symmetrischen *Sitzungsschlüssel*

Problem: nicht-manipulierbarer Kanal für Schlüsselaustausch erforderlich

Lösung: *Zertifizierung*

# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust



# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust – kann auch hierarchische Baumstruktur sein

OpenPGP: E-Mail, spezielle Anwendungen, . . .

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: persönliche Entscheidung

# 7 Verschlüsselung

## 7.2 Zertifizierung von Schlüsseln

- S/MIME: hierarchische Baumstruktur
- OpenPGP: Web of Trust – kann auch hierarchische Baumstruktur sein

OpenPGP: E-Mail, spezielle Anwendungen, ...

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: persönliche Entscheidung

S/MIME: Webseiten, E-Mail, spezielle Anwendungen, ...

- Vertrauen in den Schlüssel: mathematisch berechenbar
- Vertrauen in die Person: wird vom Anbieter vorgegeben