

Vorab: Online-Werkzeuge

- Diese Veranstaltung findet **in Präsenz** statt.
Wir versuchen aber, auch eine Online-Teilnahme zu ermöglichen.
- **Mumble**: Seminarraum 2
Fragen: Mikrofon einschalten oder über den Chat
Umfragen: über den Chat – **auch während der Präsenz-Veranstaltung**
- **VNC**: Kanal 6, Passwort: `testcvh`
Eigenen Bildschirm freigeben: VNC-Software oder Web-Interface *yesVNC*
Eigenes Kamerabild übertragen: Web-Interface *CVH-Camera*
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>
Seminarraum 2, VNC-Passwort: `testcvh`
- Bei Problemen: bitte notieren:
Art des Problems, genaue Uhrzeit, eigener Internet-Zugang
- GitLab: <https://gitlab.cvh-server.de/pgerwinski/es>

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

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

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

Was sind eingebettete Systeme?

Der Auswurf
Rechner
(eingebettete Systeme)

- keine
- in der



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

- 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

- 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

- 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

- 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-x1.c
```

- Zugriffsrechte ändern:

`chmod o-r orbit-1x.c` – Lesezugriff entziehen

`chmod g+w orbit-1x.c` – Schreibzugriff gewähren

`chmod 640 orbit-1x.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

5. Oktober 2022

Vorab: Online-Werkzeuge

- Diese Veranstaltung findet **in Präsenz** statt.
Wir versuchen aber, auch eine Online-Teilnahme zu ermöglichen.
- **Mumble**: Seminarraum 2
Fragen: Mikrofon einschalten oder über den Chat
Umfragen: über den Chat – **auch während der Präsenz-Veranstaltung**
- **VNC**: Kanal 6, Passwort: `testcvh`
Eigenen Bildschirm freigeben: VNC-Software oder Web-Interface *yesVNC*
Eigenes Kamerabild übertragen: Web-Interface *CVH-Camera*
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>
Seminarraum 2, VNC-Passwort: `testcvh`
- Bei Problemen: bitte notieren:
Art des Problems, genaue Uhrzeit, eigener Internet-Zugang
- GitLab: <https://gitlab.cvh-server.de/pgerwinski/es>

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

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

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

Was sind eingebettete Systeme?

Der Auswurf
Rechner
(eingebettet)

- keine
- in der



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

- 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 -`

```
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

- *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

- 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

- *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

12. Oktober 2022

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

...

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 -`

```
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

- *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`

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

- *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`

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

...

2.5 Pipes

2.6 Verzweigungen und Schleifen

2.7 Skript-Programmierung

2.8 GNU screen

3 TCP/IP in der Praxis

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

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

...

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 `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)

3.6 Netzwerkanalyse

- `tcpdump`
- `wireshark`
- `ettercap`

3.7 SSH

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

3.8 X11

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

Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

2. November 2022

Eingebettete Systeme

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

- 1 Einführung**
- 2 Einführung in Unix**
- 3 TCP/IP in der Praxis**
 - 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

...

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

3 TCP/IP in der Praxis

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 TCP/IP in der Praxis

3.6 Netzwerkanalyse

- `tcpdump`
- `wireshark`
- `ettercap`

3 TCP/IP in der Praxis

3.7 SSH

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

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

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

Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

9. November 2022

Eingebettete Systeme

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

1 Einführung

2 Einführung in Unix

3 TCP/IP in der Praxis

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 TCP/IP in der Praxis

3.6 Netzwerkanalyse

- tcpdump
- wireshark
- ettercap

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

Beispiel: Programmierung eines VNC-Servers per Web-Interface

- Grundlagen: RFC 6143
- Praxis: Untersuchung mit Wireshark
- Implementation: JavaScript, WebSockets, [websockify](#), Callbacks

Weitere Beispiele:

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

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg. Findet eine Datenübertragung zwischen zwei Teilnehmern statt, so müssen die übrigen Teilnehmer schweigen, da sie sich sonst gegenseitig stören würden. Umgangssprachlich werden mitunter – oft aus historischen Gründen – auch Datenübertragungssysteme als „Bus“ bezeichnet, die technisch eigentlich eine andere Topologie besitzen.

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

Beispiele:

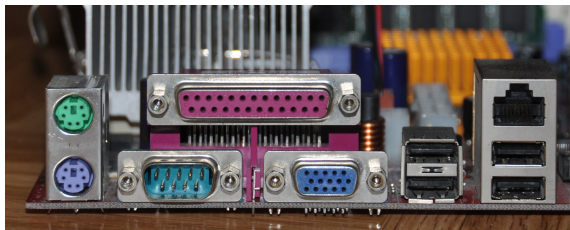
- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

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



4 Bus-Systeme

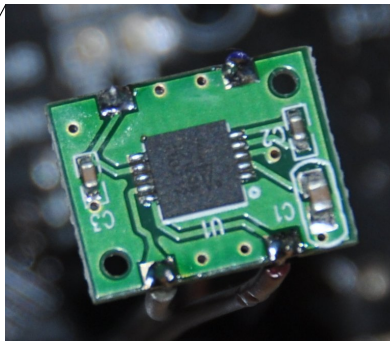
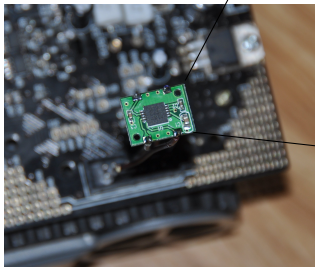
4.1 Was sind Bus-Systeme?

Standard-Personal-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
- I²C (TWI)
- SPI



4 Bus-Systeme

4.1 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>

- I²C: 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

4 Bus-Systeme

4.2 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

4.2 RS-232

Synchronisation

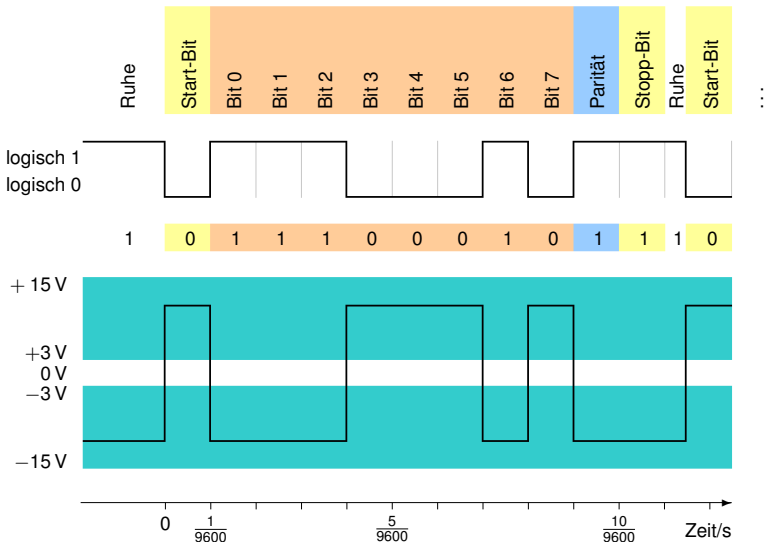
Daten

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)



4 Bus-Systeme

4.3 I²C (TWI)

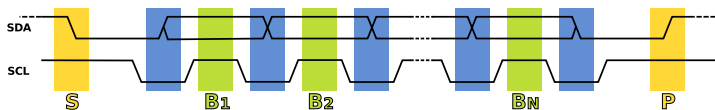
I²C = Inter-IC; TWI = Two-Wire-Interface

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

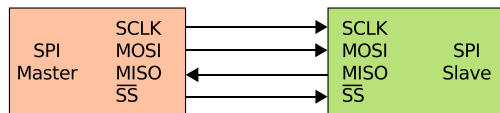
4 Bus-Systeme

4.4 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

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

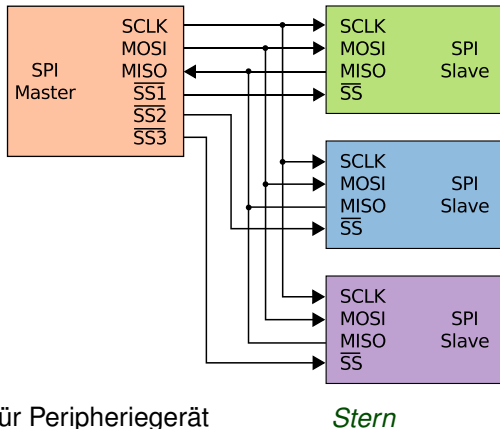
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

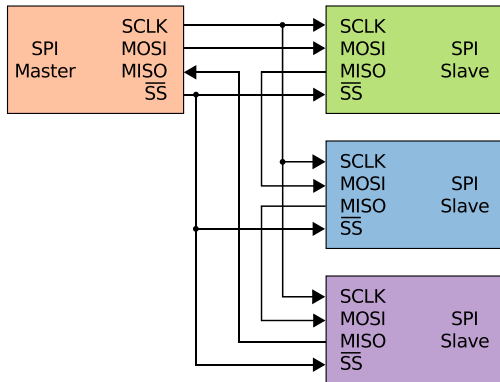
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



*Kaskade
Daisy Chain*

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

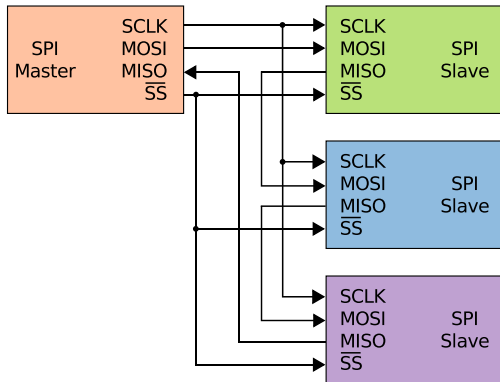
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

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}

4 Bus-Systeme

4.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

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

4 Bus-Systeme

4.6 Sonstiges

Matrix-Schaltung

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

4 Bus-Systeme

4.6 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

16. November 2022

Eingebettete Systeme

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

1 Einführung

2 Einführung in Unix

3 TCP/IP in der Praxis

...

3.6 Netzwerkanalyse

3.7 SSH

3.8 X11 und VNC

3.9 Programmierung

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

4.1 RS-232

4.1 I²C (TWI)

4.1 SPI

4.1 PWM

4.1 Sonstiges

5 Echtzeit



Änderungen
vorbehalten

3 TCP/IP in der Praxis

3.6 Netzwerkanalyse

- `tcpdump`
- `wireshark`
- `ettercap`

3 TCP/IP in der Praxis

3.9 Programmierung

Beispiel: Programmierung eines VNC-Servers per Web-Interface

- Grundlagen: RFC 6143
- Praxis: Untersuchung mit Wireshark
- Implementation: JavaScript, WebSockets, [websockify](#), Callbacks

Weitere Beispiele:

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

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg. Findet eine Datenübertragung zwischen zwei Teilnehmern statt, so müssen die übrigen Teilnehmer schweigen, da sie sich sonst gegenseitig stören würden. Umgangssprachlich werden mitunter – oft aus historischen Gründen – auch Datenübertragungssysteme als „Bus“ bezeichnet, die technisch eigentlich eine andere Topologie besitzen.

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

Beispiele:

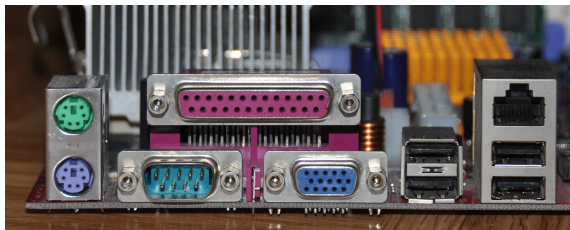
- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

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



4 Bus-Systeme

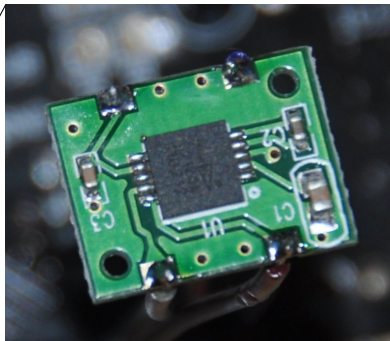
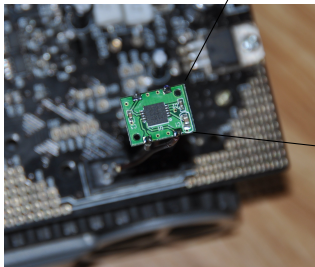
4.1 Was sind Bus-Systeme?

Standard-Personal-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
- I²C (TWI)
- SPI



4 Bus-Systeme

4.1 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>

- I²C: 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

4 Bus-Systeme

4.2 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

4.2 RS-232

Synchronisation

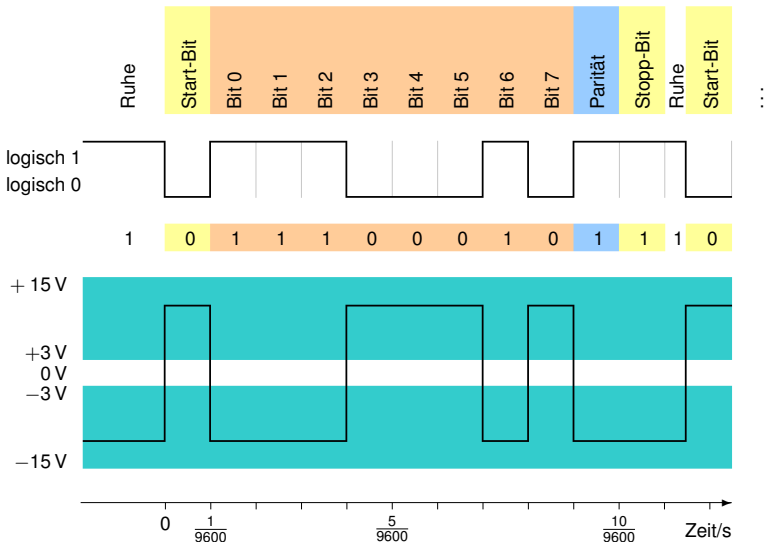
Daten

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)



4 Bus-Systeme

4.3 I²C (TWI)

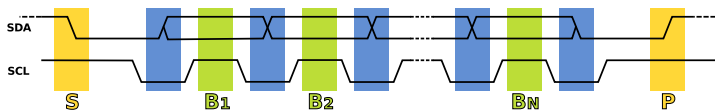
I²C = Inter-IC; TWI = Two-Wire-Interface

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

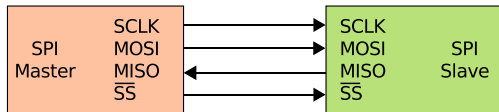
4 Bus-Systeme

4.4 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

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

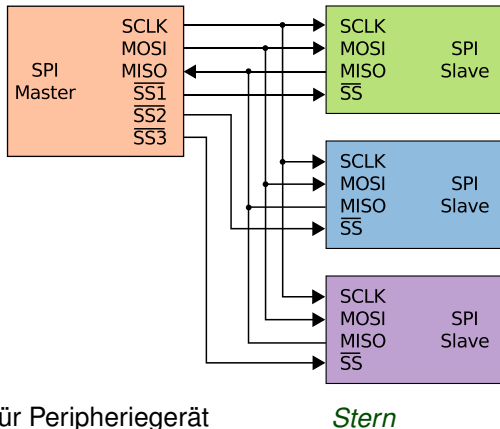
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

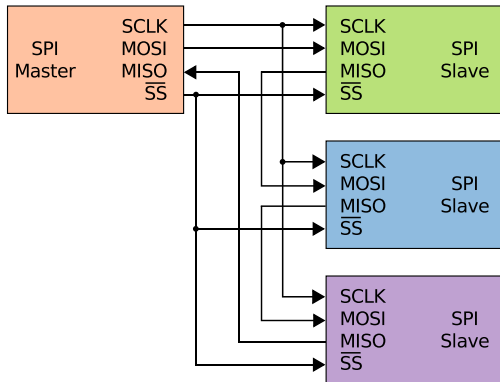
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



*Kaskade
Daisy Chain*

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

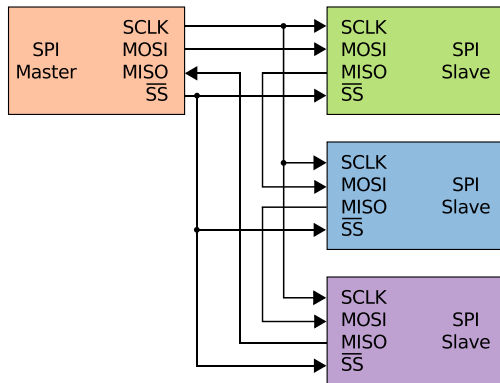
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

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}

4 Bus-Systeme

4.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

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

4 Bus-Systeme

4.6 Sonstiges

Matrix-Schaltung

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

4 Bus-Systeme

4.6 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 Bus-Systeme

4.1 Was sind Bus-Systeme?

4.1 RS-232

4.1 I²C (TWI)

4.1 SPI

4.1 PWM

4.1 Sonstiges

5 Echtzeit

5.1 Was ist Echtzeit?

5.2 Echtzeitprogrammierung

5.3 Multitasking

5.4 Ressourcen

5.5 Prioritäten



Änderungen
vorbehalten

Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

23. November 2022

Eingebettete Systeme

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

1 Einführung

2 Einführung in Unix

3 TCP/IP in der Praxis

4 Bus-Systeme

4.1 Was sind Bus-Systeme?

4.2 RS-232

4.3 I²C (TWI)

4.4 SPI

4.5 PWM

4.6 Sonstiges

5 Echtzeit

5.1 Was ist Echtzeit?

5.2 Echtzeitprogrammierung

5.3 Multitasking

5.4 Ressourcen

5.5 Prioritäten



Ergänzungen
noch möglich

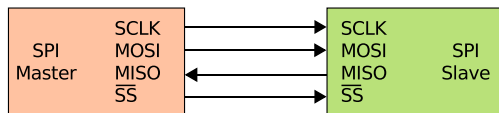
4 Bus-Systeme

4.4 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

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

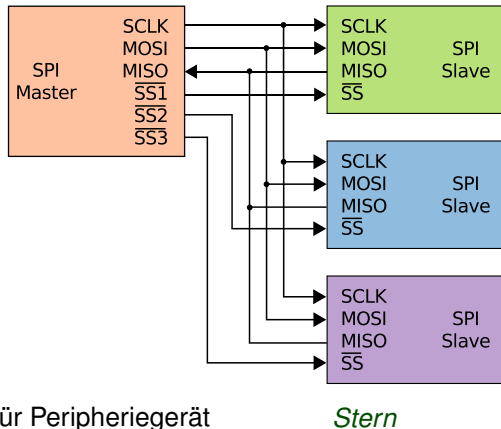
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

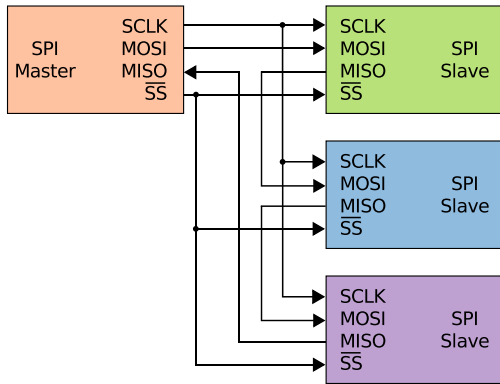
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

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



*Kaskade
Daisy Chain*

4 Bus-Systeme

4.4 SPI

Serial Peripheral Interface

seriell

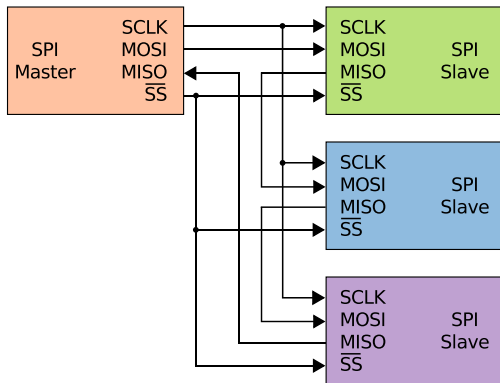
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- \overline{SS} : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

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}

4 Bus-Systeme

4.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

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

4 Bus-Systeme

4.6 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 Bus-Systeme

4.1 Was sind Bus-Systeme?

4.2 RS-232

4.3 I²C (TWI)

4.4 SPI

4.5 PWM

4.6 Sonstiges

5 Echtzeit

5.1 Was ist Echtzeit?

5.2 Echtzeitprogrammierung

5.3 Multitasking

5.4 Ressourcen

5.5 Prioritäten



Ergänzungen
noch möglich

5 Echtzeit

5.1 Was ist Echtzeit?

5 Echtzeit

5.1 Was ist Echtzeit?

- Animation in Echtzeit:
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung

5 Echtzeit

5.1 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.

5 Echtzeit

5.1 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.

5 Echtzeit

5.1 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.

5 Echtzeit

5.1 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.“

5.1 Was ist Echtzeit?

„Schnell genug.“

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

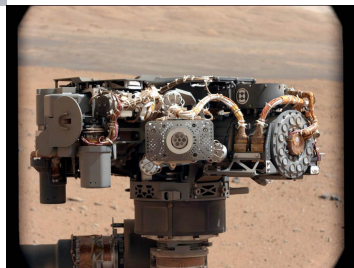
- „Ganz schlecht.“ → *harte Echtzeit*



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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



5.1 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

5.1 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.“

5.1 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*

5.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

5.1 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.

5.1 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

5.1 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

5.1 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?

5.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

„Verschwendung von Rechenzeit – na und?“

5.1 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

5.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

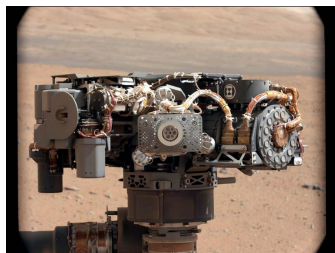
„Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

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

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



5.1 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



5.1 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

5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

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

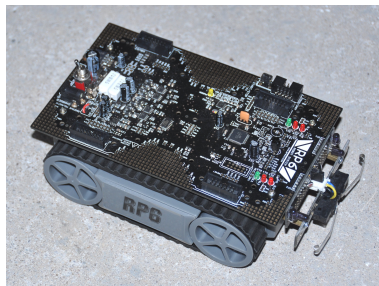


5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

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



5.2 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



5.2 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



5.2 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.

5.2 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 ± 1 -Schritten

5.2 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 ± 1 -Schritten
- Nebenbei: Benutzerprogramm

5.2 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 ± 1 -Schritten
- Nebenbei: 1 Benutzerprogramm

5.3 Multitasking

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

5.3 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

5.3 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

Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

7. Dezember 2022

Eingebettete Systeme

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

- 1 Einführung
- 2 Einführung in Unix
- 3 TCP/IP in der Praxis
- 4 Bus-Systeme
- 5 Echtzeit
 - 5.1 Was ist Echtzeit?
 - 5.2 Echtzeitprogrammierung
 - 5.3 Multitasking
 - 5.4 Ressourcen
 - 5.5 Prioritäten



Ergänzungen
noch möglich

5 Echtzeit

5.1 Was ist Echtzeit?

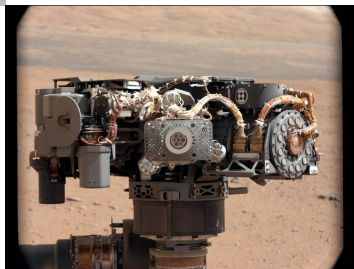
- Animation in Echtzeit:
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung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

→ „Schnell genug.“

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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

5.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

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



5.1 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

5.1 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*

5.1 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?

5.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

„Verschwendung von Rechenzeit – na und?“

5.1 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

5.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

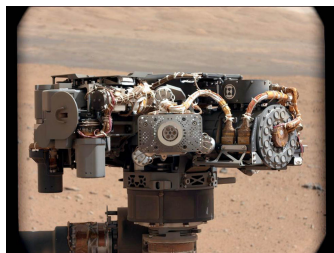
„Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

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

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



5.1 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



5.1 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

5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

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

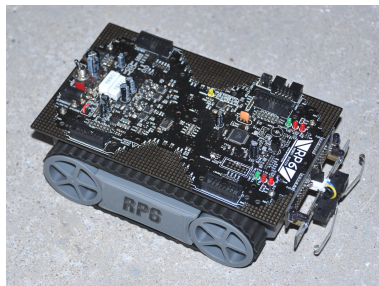


5.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

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



5.2 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



5.2 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



5.2 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.

5.2 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 ± 1 -Schritten

5.2 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 ± 1 -Schritten
- Nebenbei: Benutzerprogramm

5.2 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 ± 1 -Schritten
- Nebenbei: 1 Benutzerprogramm

5.3 Multitasking

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

5.3 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

5.3 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.

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

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

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:

Latenz vs. *Jitter*

vs. *Durchsatz*

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

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

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

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

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

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

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

5 Echtzeit

5.3 Multitasking

Qualitätsaspekte beim Multitasking

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

5 Echtzeit

5.3 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, ...)

5 Echtzeit

5.3 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)

5 Echtzeit

5.4 Ressourcen

Ressourcen reservieren

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

5 Echtzeit

5.4 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

5 Echtzeit

5.4 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

5 Echtzeit

5.4 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

5 Echtzeit

5.4 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 beanspruchen, notfalls warten
Zu Zeile 3.7: c1.acquire(1) ist bereits lock-entriegelt, ab Zeile 62

void __sched_mutex_lock(struct mutex *lock)
{
    might_sleep();
    __mutex_lock(lock, (lock->owner == NULL) ? __mutex_lock_always :
    __mutex_lock_owner);
}

Code Zeile 3.7: c1.acquire(1) ist bereits lock-entriegelt, ab Zeile 24
Kern-Definition für __mutex_lock(lock, expandier)
{
    lock_desc(desc->lock) = current;
    __pw 1
    c1:
        __mutex_lock_always

Zu Zeile 3.7: c1.acquire(1) ist bereits lock-entriegelt, ab Zeile 308
static inline void __sched
__mutex_lock_owner(struct mutex *lock, int count)
{
    struct mutex *lock = container_of(lock_desc, struct mutex, count);
    __mutex_lock_common(lock, TASK_UNINTERRUPTIBLE, 0,
    NULL, RET_P);
}

Code Zeile 3.7: c1.acquire(1) ist bereits lock-entriegelt, ab Zeile 132
static inline int __sched
__mutex_lock_common(struct mutex *lock, long unsigned int subsecs,
    struct task_struct *task, unsigned long ip)
{
    struct task_struct *current;
    struct mutex *waiter;
    unsigned long long;

    preempt_disable();
    __mutex_acquire_wait(lock->dead_map, subsecs, 0, next_lock, ip);
    //

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

    // add waiting tasks to the end of the waitqueue (FIFO):
    list_add_tail(&waiter, lock->wait_list);
    waiter_lock = task;

    if (atomic_cmp(&lock->owner, -1) == 1)
        goto done;

    lock_contended(lock->dead_map, ip);

    for (;;)
    {
        // Let's 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 wake-up once if
        // the unlocker calls the TASK_INTERRUPTIBLE case.
        // The operation that gave us the lock. We use it to – 1, as
        // that's when we release the lock. We properly wake up the
        // other waiters.
        if (atomic_cmp(&lock->owner, -1) == 1)
            break;

        // got a signal? (this code gets eliminated in the
        // TASK_INTERRUPTIBLE case.)
        if (unlikely(jumping_gdang_task_state(task)))
        {
            mutex_remove_waiter(lock, &waiter, task, thread_info(task));
            mutex_release(lock->dead_map, 1, ip);
            spin_unlock_mutex(lock->wait_lock, flag);
            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, flag);
        schedule_preempt_disabled();
        spin_lock_mutex(lock->wait_lock, flag);

    done:
        lock_acquired(lock->dead_map, ip);
        // got the lock – myself:
        mutex_remove_waiter(&waiter, current, thread_info(task));
        __mutex_unlock(lock);
        atomic_set(&lock->owner, 0);

    spin_unlock_mutex(lock->wait_lock, flag);
    debug_mutex_free_waiter(&waiter); // exklusiven Zugriff auf Mutex wieder freigeben
    preempt_enable();

    return 0;
}

```


spin_lock_mutex() - Mutex beanspruchung, notfalls busy waiting
 Datei linux-3.7.rc1/kernel/mutex.h, ab Zeile 12

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

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

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

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

```
#define raw_spin_lock(lock) __raw_spin_lock(lock)
```

Datei linux-3.7.rc1/include/linux/spinlock_api_smp.h, Zeile 47

```
#define __raw_spin_lock(lock) ___raw_spin_lock(lock)
```

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

```
void __lockfunc ___raw_spin_lock (spinlock_t *lock) \
{ \
    for (;;) \
    { \
        preempt_disable (); \
        if (likely (do_raw_spin_trylock (lock))) \
            break; \
        preempt_enable (); \
        if (!lock) --> break; \
        (lock) --> break; \
        while (raw_spin_can_lock (lock) && (lock) --> break; \
        arch_spin_relax (&lock->raw_lock); \
    } \
    (lock) --> break; \
}
```

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

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

Datei arch/x86/include/asm/spinlock.h, ab Zeile 116:

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

Datei arch/x86/include/asm/spinlock.h, ab Zeile 65:

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

Datei arch/x86/include/asm/cmpchg.h, ab Zeile 147:

```
#define cmpchg(ptr, old, new, sizeof) \
    __cmpchg (ptr, old, new, sizeof)
```

Datei arch/x86/include/asm/cmpchg.h, ab Zeile 131:

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

5 Echtzeit

5.4 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

5 Echtzeit

5.4 Ressourcen

Verklemmungen: Gegenseitiges Blockieren von Ressourcen

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

5 Echtzeit

5.4 Ressourcen

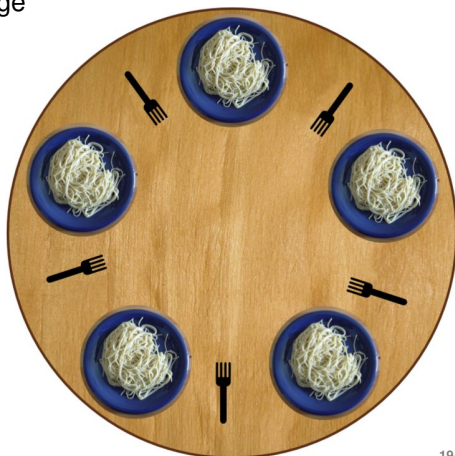
Verklemmungen: Gegenseitiges Blockieren von Ressourcen

- **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**



5 Echtzeit

5.4 Ressourcen

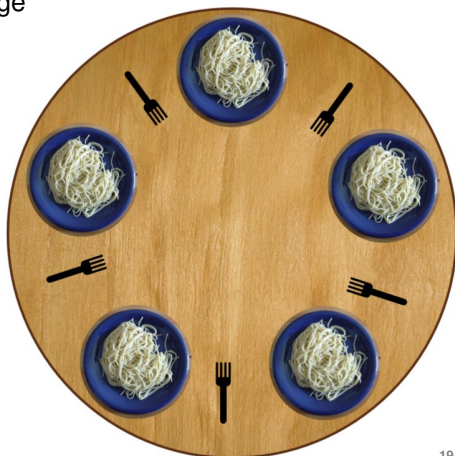
Verklemmungen: Gegenseitiges Blockieren von Ressourcen

- **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**



5 Echtzeit

5.4 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

5 Echtzeit

5.4 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 |

5 Echtzeit

5.5 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.

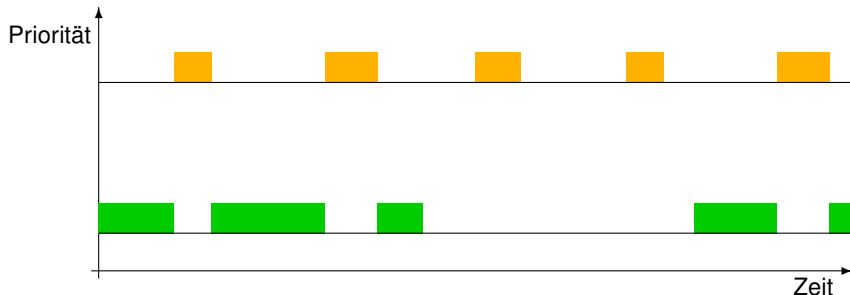
→ *statische Prioritätenvergabe*

5 Echtzeit

5.5 Prioritäten

Echtzeitbetriebssysteme

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

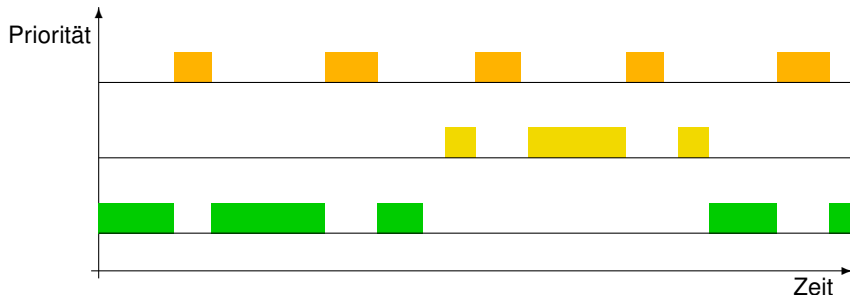


5 Echtzeit

5.5 Prioritäten

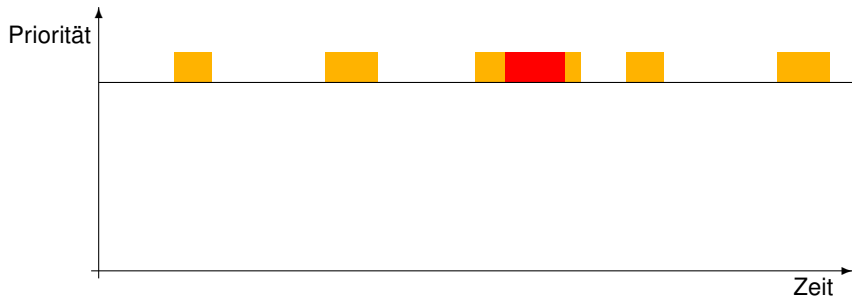
Echtzeitbetriebssysteme

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



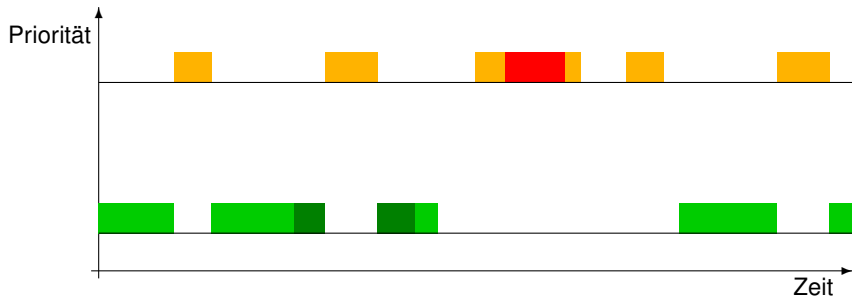
5 Echtzeit

5.5 Prioritäten und Ressourcen



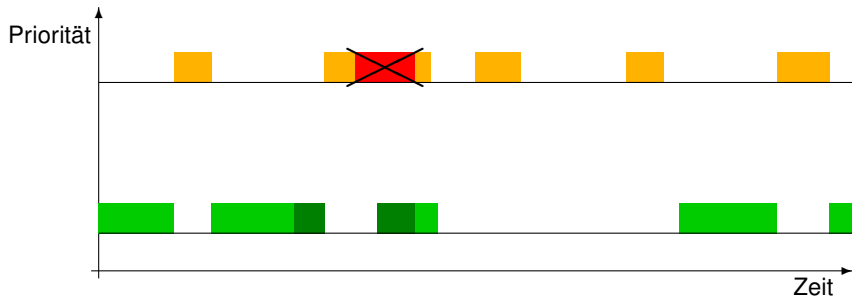
5 Echtzeit

5.5 Prioritäten und Ressourcen



5 Echtzeit

5.5 Prioritäten und Ressourcen



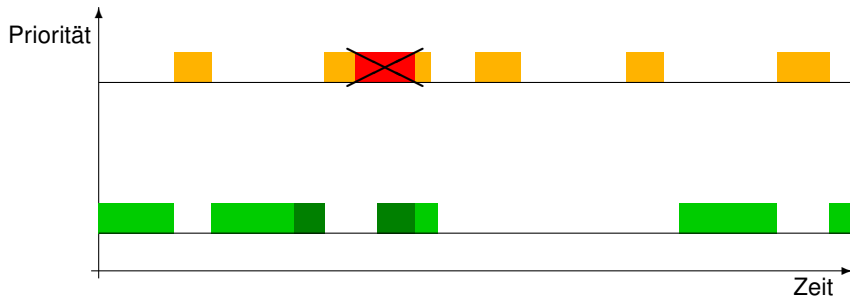
5 Echtzeit

5.5 Prioritäten und Ressourcen

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

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



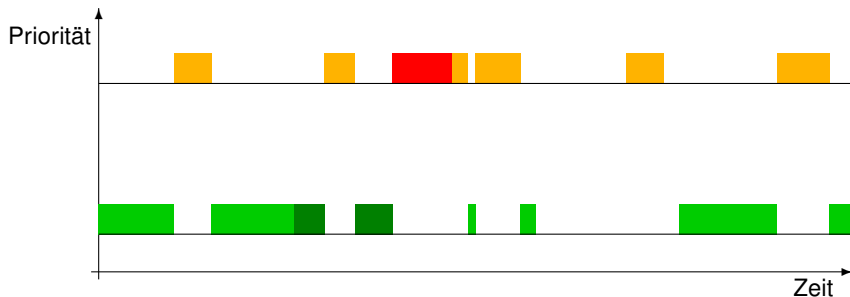
5 Echtzeit

5.5 Prioritäten und Ressourcen

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

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



5 Echtzeit

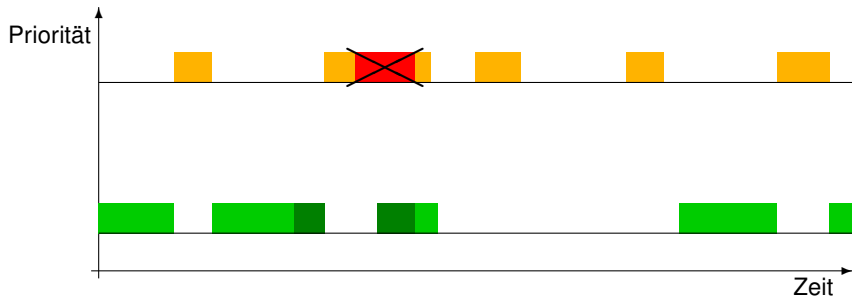
5.5 Prioritäten und Ressourcen

unbegrenzte Prioritätsinversion

5 Echtzeit

5.5 Prioritäten und Ressourcen

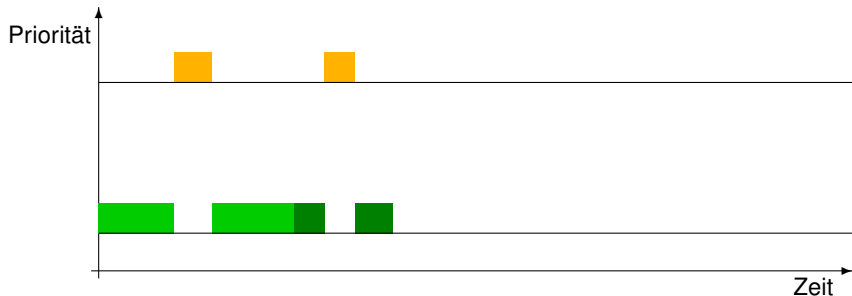
unbegrenzte Prioritätsinversion



5 Echtzeit

5.5 Prioritäten und Ressourcen

unbegrenzte Prioritätsinversion

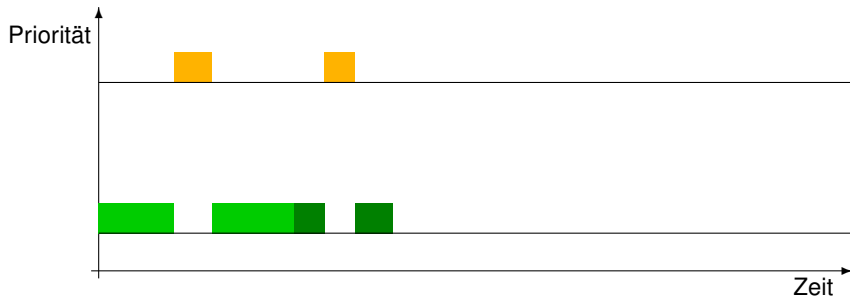


5 Echtzeit

5.5 Prioritäten und Ressourcen

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

→ *unbegrenzte Prioritätsinversion*

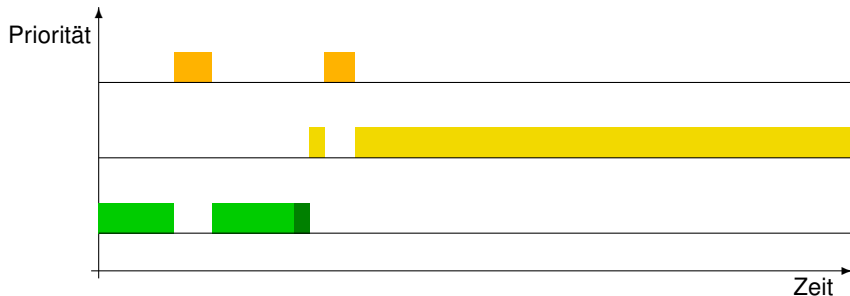


5 Echtzeit

5.5 Prioritäten und Ressourcen

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

→ *unbegrenzte Prioritätsinversion*



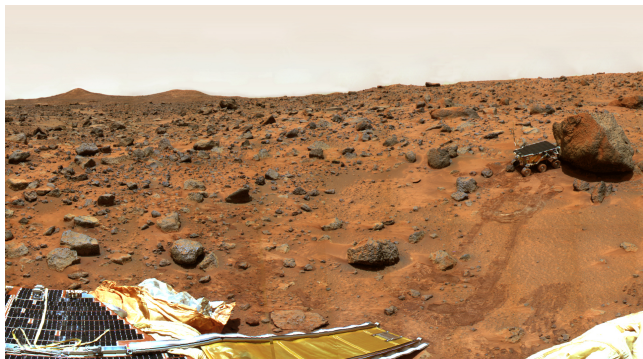
5 Echtzeit

5.5 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



5 Echtzeit

5.5 Prioritäten und Ressourcen

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

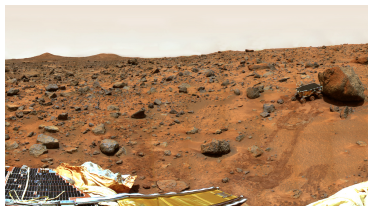
—→ *unbegrenzte Prioritätsinversion*

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/



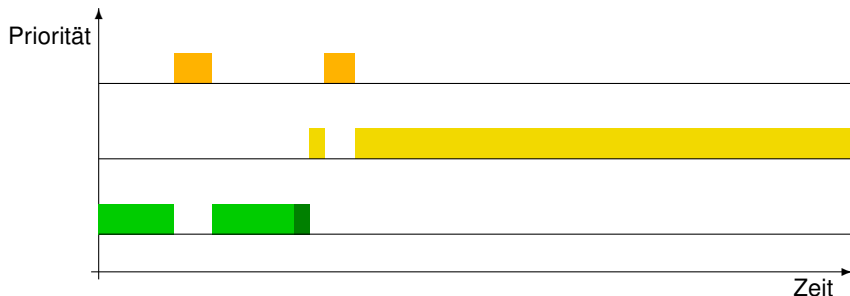
5 Echtzeit

5.5 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*



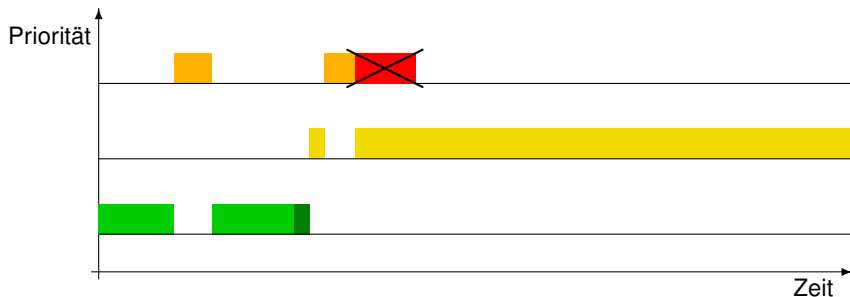
5 Echtzeit

5.5 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*



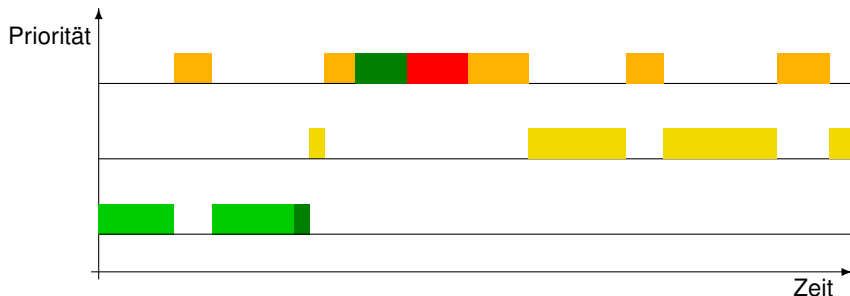
5 Echtzeit

5.5 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*



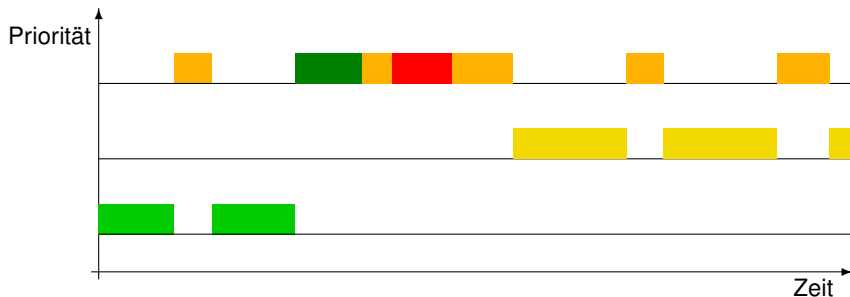
5 Echtzeit

5.5 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*



5 Echtzeit

5.5 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.

5 Echtzeit

5.5 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

5 Echtzeit

5.5 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