

System-Programmierung

0: Einführung

CC BY-SA, Thomas Amberg, FHNW
(soweit nicht anders vermerkt)

Ablauf heute

$\frac{1}{3}$ Vorlesung,

$\frac{2}{3}$ Hands-on,

Feedback.

Slides, Code & Hands-on: tmb.gr/syspr-0



Hallo

Thomas Amberg ([@tamberg](#)), Software Ingenieur.

Neu an der FHNW als Prof. für Internet of Things.

Gründer von [Yaler](#), "sicherer Fernzugriff für IoT".

Organisator [IoT Meetup](#), [Maker Faire](#) in Zürich.

thomas.amberg@fhnw.ch

Aufbau Modul *syspr*

$15 * 3 = 45$ Stunden Unterricht:

$\frac{1}{3}$ Vorlesung plus $\frac{2}{3}$ Hands-on.

Dazu ca. 45 Stunden Selbststudium.

Total 90 Stunden, d.h. 3 ECTS Punkte.

Lernziele Modul *syspr*

Programmierung in C, da der Unix/Linux-Kern und Basisanwendungen in der Sprache geschrieben sind.

Praktische Nutzung der System-Call Schnittstelle von Unix/Linux lernen anhand von Beispielprogrammen.

Kommunikation zwischen Prozessen (IPC) und deren Synchronisation verstehen und einsetzen lernen.

Termine 2019 — Klasse 4ibb1

| | | | | |
|--------|----------------------|--------|-----------------------|--------|
| 19.02. | Einführung | 16.04. | IPC mit Pipes | Ferien |
| 26.02. | Erste Schritte in C | 30.04. | Sockets | |
| 05.03. | Funktionen | 07.05. | (Projektwoche) | |
| 12.03. | File In-/Output | 14.05. | Terminals | |
| 19.03. | Prozesse und Signale | 21.05. | POSIX IPC | |
| 26.03. | Prozess-Lebenszyklus | 28.05. | Zeitmessung | |
| 02.04. | Threads und Synchr. | 04.06. | Assessment II | |
| 09.04. | Assessment I | 11.06. | Weitere Arten von I/O | |

Termine 2019 — Klasse 4ibb2

18.02. Einführung
25.02. Erste Schritte in C
04.03. Funktionen
11.03. File In-/Output
18.03. Prozesse und Signale
25.03. Prozess-Lebenszyklus
01.04. Threads und Synchr.
08.04. Assessment I

15.04. IPC mit Pipes
29.04. Sockets
06.05. (Projektwoche)
13.05. Terminals
20.05. POSIX IPC
27.05. Zeitmessung
03.06. Assessment II
10.06. (Pfingstmontag)

Ferien

Lernzielüberprüfung

Zwei obligatorische Assessments, Woche 15 und 23.

Fliessen zu je 50% in die Gesamtbewertung ein.

Die Schlussnote wird auf Zehntel gerundet.

Es gibt keine Modulschlussprüfung.

Assessments

C-Referenzkarte wird vom Dozenten verteilt.

Weitere Unterlagen sind nicht erlaubt.

Die Assessments sind schriftlich.

Und dauern je 90 Minuten.

Beispiele vom HS18: [Assessment I](#), [Assessment II](#). 

Betrug und Plagiate

Aus **Betrug und Plagiate bei Leistungsnachweisen:**

"Wer in Arbeiten im Rahmen des Studiums Eigen- und Fremdleistung nicht unterscheidet, wer plagiiert, macht sich strafbar." - M. Meyer

Fremden Code kennzeichnen, Lizenzen beachten. 

Unterricht

Slides, Code und Hands-on sind Prüfungsstoff.

Slides als PDF, Code-Beispiele sind verlinkt.

Hands-on laufend, via GitHub abgeben.

Review? GitHub Issue, @tamberg.

Hands-on Sessions

"Be excellent to each other", Fragen / Helfen ist OK.

Google ([DDG.co](https://duckduckgo.com/), ...) nutzen um Fehler zu beheben.

Blind kopieren bringt keine neuen Einsichten.

Fremden, guten Code lesen hingegen schon.

Ablage Slides, Code & Hands-on

<http://tmb.gr/syspr> →

<https://github.com/tamberg/fhnw-syspr>

01/

hello.c

README.md → Slides, Hands-on

02/

...

Abgabe Hands-on Resultate via GitHub

<https://github.com/fhnw-syspr-4ibb1> bzw. [-4ibb2](https://github.com/fhnw-syspr-4ibb2)

| | |
|-------------------------|-------------------------|
| fhnw-syspr-work-01 | Repo Vorlage mit Link |
| fhnw-syspr-work-01-USER | Repo Kopie pro User |
| README.md | Hands-on Aufgaben |
| my_result.c | "Privat", Dozent & User |

Wieso GitHub? Professionelles Tool, zugleich Backup.

Wieso Repo/Lektion? Einfacher als Forks updaten.

Kommunikation mit Slack

<https://fhnw-syspr.slack.com/>

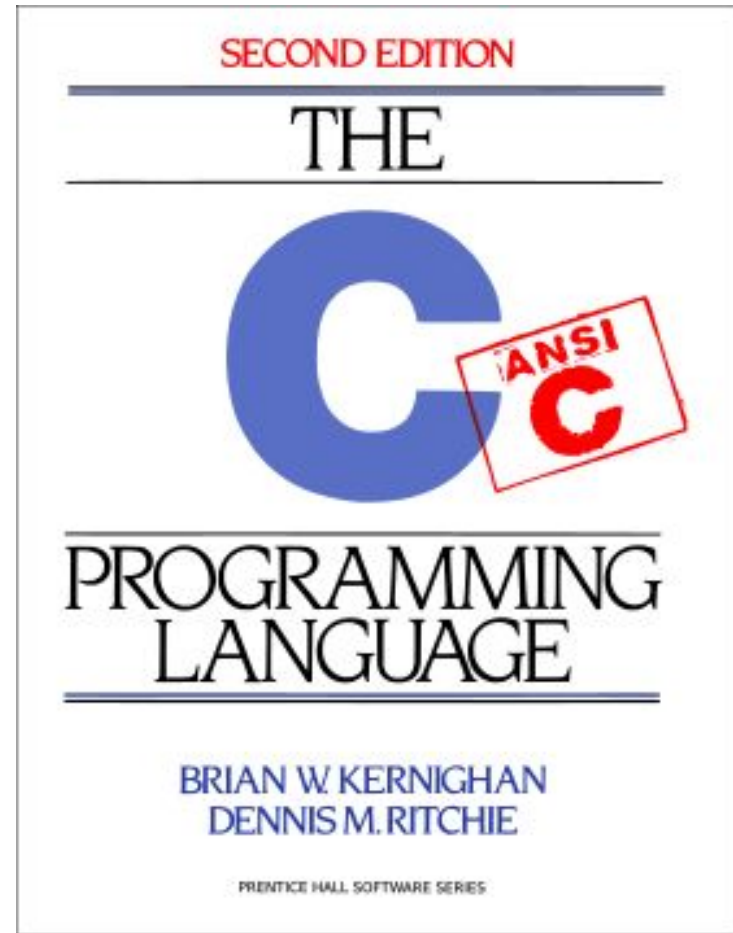
| | |
|-----------|------------------------------------|
| #general | Allg. Fragen und Ankündigungen. |
| #random | Eher Unwichtiges, Zufälliges. |
| #c-lang | C spezifische Fragen. |
| # ... | Weitere Channels. |
| • tamberg | Messages an eine Person, "privat". |

Slack App wird empfohlen, mobile oder Desktop.

Literatur

<https://ddg.co/?q=the+c+programming+language+kernighan+ritchie>

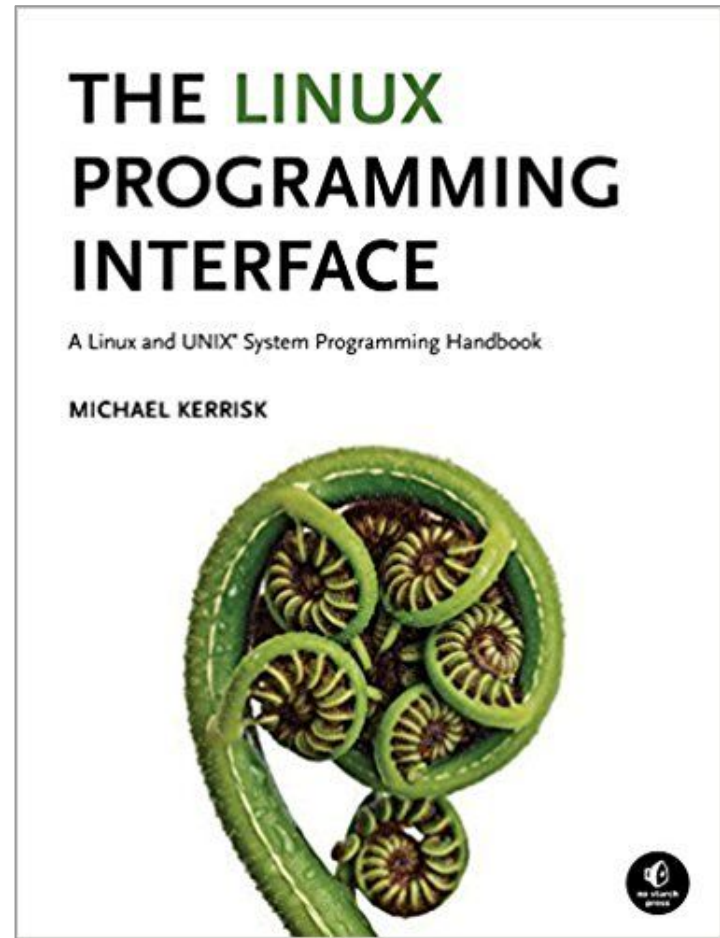
Klassiker, 270 Seiten.



Literatur (optional)

<https://ddg.co/?q=the+linux+programming+interface>

Nachschlagwerk,
1500+ Seiten.



Tools

Terminal (MacOS) bzw. *cmd* (Windows).

Text-Editor, z.B. *nano* oder **VS Code**.

C Compiler, *gcc* / Debugger, *gdb*.

Code Versionierung mit *git*.

Einfache Tools, ohne "Magie" => Verständnis.

Linux, VM oder Raspberry Pi

System-Programmierung am Beispiel von Linux.

Die Code-Beispiele sind auf Raspbian getestet.

Im Prinzip sollte der C Code portabel sein.

Sie können auch eine VM verwenden.

Wieso Raspberry Pi?

Günstige Hardware.

Einheitliche Linux Plattform.

Separates System => "Sandbox".

SD Card neu schreiben => "Factory reset".

Embedded Linux Systeme sind relevant für IoT.

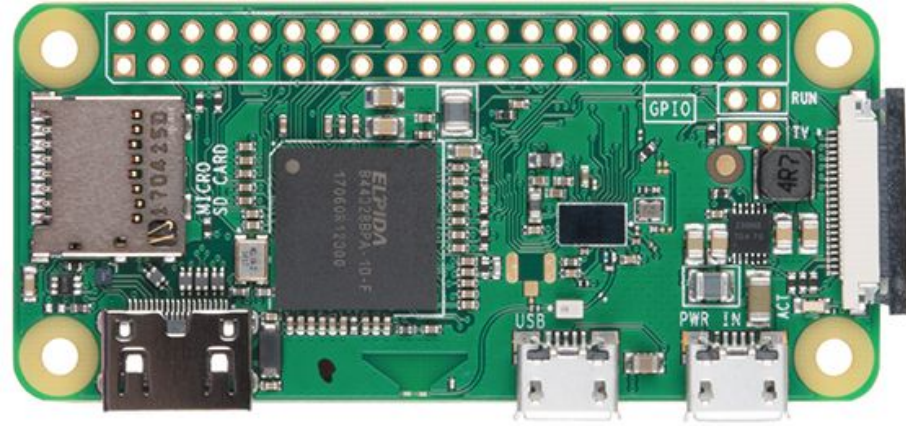
Raspberry Pi

Einplatinencomputer

[https://raspberrypi.org/
products/raspberry-pi-zero-w/](https://raspberrypi.org/products/raspberry-pi-zero-w/)

1GHz, single core ARM CPU, 512 MB RAM,
Mini HDMI, USB On-The-Go, Wi-Fi, Bluetooth, etc.

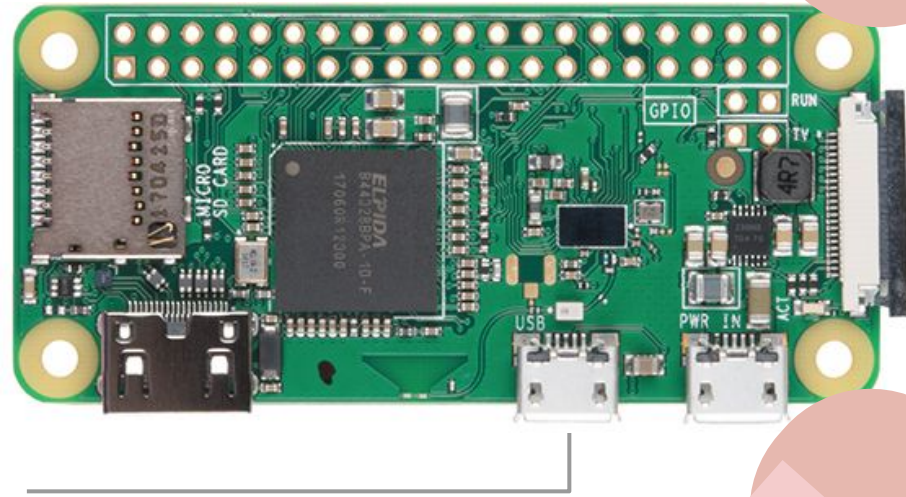
Leihweise, inklusive USB Kabel, gegen Unterschrift.



Raspberry Pi Setup

Raspbian "Stretch Lite"
Linux IMG auf SD Card.

Internet-Sharing via USB.



Getestet auf MacOS und Windows 10.

Raspberry Pi SD Card erstellen

[Etcher](#) Tool installieren, [Raspbian](#) "Stretch Lite" IMG herunterladen und mit Etcher auf leere SD Card spielen.

(IMG Datei auf SD Card kopieren geht nur mit Tool.)

Um SSH einzuschalten, leere Datei `ssh` erstellen:

MacOS, Linux:

```
$ cd /Volumes/boot
```

```
$ touch ssh
```

Windows:

```
C:\> E:
```

```
E:\> type nul > ssh
```

Raspberry Pi Zero W als USB Gadget

Auf SD Card in *config.txt* neue Zeile *dtoverlay=dwc2*:

```
$ open config.txt
```

...

```
dtoverlay=dwc2
```

In *cmdline.txt* nach *rootwait* diesen Text einfügen:

```
$ open cmdline.txt
```

```
... rootwait modules-load=dwc2,g_ether ...
```

(Windows: *open* durch *notepad* ersetzen.)

Internet-Sharing von Wi-Fi zu USB

MacOS

System Preferences > Sharing > [✓] Internet Sharing > Share your connection from: Wi-Fi to computers using RNDIS Ethernet Gadget

Windows (vorher [Bonjour installieren](#))

WINDOWS-R > ncpa.cpl > CTRL-Klick Wi-Fi und RNDIS Ethernet adapter > Rechtsklick > Bridge

Oder Wi-Fi > Properties > Sharing > [✓] Allow

Wi-Fi Konfiguration zu Hause (optional)

Auf Raspi, Datei *wpa_supplicant.conf* ergänzen, mit:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

...

```
network={  
    ssid="WIFI_SSID"  
    psk="WIFI_PASSWORD"  
    key_mgmt=WPA-PSK  
}
```

Oder via SD Card, in */boot/wpa_supplicant.conf* 

Wi-Fi Konfiguration für fhnw-private

```
$ echo -n 'PASSWORD' | iconv -t utf16le | openssl md4  
=> PW_HASH, e.g. 62f6e1dc44a0eac6784f134e1c2c2b03  
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

```
network={  
    ssid="fhnw-private"  
    scan_ssid=1  
    priority=1  
    proto=RSN  
    key_mgmt=WPA-EAP  
    pairwise=CCMP
```

```
}
```

```
    auth_alg=OPEN  
    eap=PEAP  
    identity="FHNW_EMAIL"  
    password=hash:PW_HASH  
    phase1="peaplabel=0"  
    phase2="auth=MSCHAPV2"
```

Zugriff auf den Raspberry Pi mit SSH

Auf Windows mit dem **PuTTY** Tool:

Host: raspberrypi.local, Port: 22, User: pi

Auf MacOS und Linux mit *ssh*:

```
$ ssh pi@raspberrypi.local
```

Oder *ssh* mit IP Adresse, z.B.

```
$ ssh pi@192.168.0.42
```

```
pi@192.168.0.42's password: raspberry
```

Raspberry Pi finden im lokalen Netzwerk

IP Adresse finden, auf MacOS und Linux mit *dns-sd*:

```
$ dns-sd -G v4 raspberrypi.local
```

Oder mit *ifconfig* (bzw. *ipconfig*) und **nmap**:

```
$ ifconfig
```

```
en0: ... inet 192.168.0.23
```

```
$ nmap 192.168.0.0-255 -p 22
```

```
Nmap scan report for 192.168.0.42
```

```
22/tcp open  ssh
```

Achtung:
Keine Port
Scans an
der FHNW!

Linux/Unix Shell Kommandos

| | |
|-----------------------------|----------------------------|
| \$ ls | <i>Directory auflisten</i> |
| \$ mkdir my_directory | <i>Directory erstellen</i> |
| \$ cd my_directory | <i>Directory öffnen</i> |
| \$ echo "my file" > my_file | <i>(Datei erstellen)</i> |
| \$ cat my_file | <i>Datei anzeigen</i> |
| \$ rm my_file | <i>Datei löschen</i> |
| \$ man rm | <i>Doku zu rm anzeigen</i> |

Mehr [hier](#) oder auf [tldr.sh](#) (auch als [PDF](#)).

Textdatei erstellen auf dem Raspberry Pi

Copy & Paste in eine neue Datei *hello.c*:

```
$ nano hello.c {Text einfügen}
```

Speichern und *nano* beenden:

```
CTRL-X Y ENTER
```

Anzeigen der Datei:

```
$ cat hello.c
```

Datei kopieren zum/vom Raspberry Pi

Auf Windows mit dem **WinSCP** Tool.

Auf MacOS oder Linux mit **FileZilla** oder *scp*.

Datei vom Computer zum Raspberry Pi kopieren:

```
$ scp -P 22 LOCAL_FILE pi@RASPI_IP:RASPI_PATH
```

Bzw. vom Raspberry Pi auf den Computer kopieren:

```
$ scp -P 22 pi@RASPI_IP:RASPI_FILE LOCAL_PATH
```


Datei runterladen auf den Raspberry Pi

Datei runterladen mit *wget*:

```
$ wget -O LOCAL_PATH REMOTE_URL
```

```
$ wget -O hello.c https://raw.githubusercontent.com/leachim6/hello-world/master/c/c.c
```

Oder, wenn der Ziel-Dateiname identisch ist:

```
$ wget https://raw.githubusercontent.com/antirez/kilo/master/kilo.c
```

Hands-on, 30': Raspberry Pi

*Grundlage für das
ganze Modul syspr.*

Raspberry Pi Setup via USB zum eigenen Computer.

"Hello World" in C auf Raspberry Pi speichern.

Den C Source Code mit *gcc* kompilieren.

```
$ gcc -o hello hello.c
```

```
$ ./hello
```

Fertig? Bitte Nachbarn helfen.

Source Code Versionierung mit Git

Account erstellen auf [GitHub.com](https://github.com).

=> USER_NAME, USER_EMAIL

Auf dem Raspberry Pi, *git* installieren mit *apt-get*:

```
$ sudo apt-get update
```

```
$ sudo apt-get install git
```

Installation prüfen:

```
$ git
```

Git konfigurieren auf dem Raspberry Pi

User konfigurieren:

```
$ git config --global user.email "USER_EMAIL"  
$ git config --global user.name "USER_NAME"
```

SSH Key erstellen:

```
$ ssh-keygen -t rsa -b 4096 -C "USER_EMAIL"  
$ eval "$(ssh-agent -s)"  
$ cat ~/.ssh/id_rsa.pub
```

GitHub konfigurieren

Raspberry Pi **SSH Key eintragen** auf **GitHub**:

User Icon > Settings > SSH and GPG keys > New SSH key > {SSH Key einfügen}

Auf Raspberry Pi, Passphrase ablegen in *keychain*:

```
$ sudo apt-get install keychain
```

```
$ keychain ~/.ssh/id_rsa
```

```
$ . ~/.keychain/$HOSTNAME-sh
```

(Bei Reboot wird der *keychain* Cache gelöscht.)

GitHub Repository klonen

(GitHub Repository [erstellen](#).)

GitHub Repository klonen:

```
$ git clone git@github.com:USER_NAME/REPO.git
```

Neue Datei hinzufügen:

```
$ cd REPO
```

```
$ nano my.c
```

```
$ git add my.c
```

Git verwenden auf dem Raspberry Pi

Geänderte Dateien anzeigen:

```
$ git status
```

Änderungen committen:

```
$ git commit -a -m "fixed all bugs"
```

Änderungen pushen:

```
$ git push
```

Mehr zu *git* [hier](#).

Hands-on, 30': GitHub

*Grundlage für das
ganze Modul syspr.*

GitHub Account einrichten, falls keiner vorhanden.

Git (auf Raspberry Pi) installieren & konfigurieren.

Hands-on Repo erzeugen aus [/fhnw-syspr-work-00](#)

D.h. dem Link folgen => Forks => Classroom Link.

Dann das Hands-on Repo (auf Raspberry Pi) klonen.

File hello.c in Hands-on Repo committen, pushen. 

Selbststudium, 3h: Pointers and Arrays

Lesen Sie die [C-Referenzkarte](#) als Übersicht zu C.

Als Vorbereitung auf die nächste Lektion, lesen Sie [\[K&R\]](#) *Chapter 5: Pointers and Arrays* bis p.126.

Die nächste Lektion fasst dann beides zusammen, ohne Selbststudium wird das Tempo eher hoch sein.

Feedback?

Gerne im [Slack](#) oder an thomas.amberg@fhnw.ch

Programmierfragen am besten schriftlich.

Sprechstunde auf Voranmeldung.

Slides, Code & Hands-on: tmb.gr/syspr-0

