

# System-Programmierung

## 0: Einführung (30.03.2020)

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

# Ablauf heute

1/2 Einführung,

1/2 Hands-on.

Slides, Code & Hands-on: [tmb.gr/syspr-0](https://tmb.gr/syspr-0)



# Hallo

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

FHNW seit 2018 als "Prof. für Internet of Things".

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

Organisator der [IoT Meetup](#) Gruppe in Zürich.

[thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

# Aufbau Modul *syspr*

15 \* 3 = 45 Stunden Unterricht:

Hands-on während der Lektion.

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 FS20 — Klasse 4ibb1

		Ferien
18.02.	Einführung	21.04. IPC mit Pipes
25.02.	Erste Schritte in C	28.04. Sockets
03.03.	Funktionen	05.05. (Selbststudium)
10.03.	File In-/Output	12.05. POSIX IPC
17.03.	Prozesse und Signale	19.05. Zeitmessung
24.03.	Prozesse und Signale	26.05. Terminals
31.03.	Prozess-Lebenszyklus	02.06. <b>Assessment</b>
07.04.	Threads und Synchr.	09.06. Weitere Arten von I/O

# Termine FS20 — Klasse 4ibb2

20.02. Einführung  
27.02. Erste Schritte in C  
05.03. Funktionen  
12.03. File In-/Output  
19.03. Prozesse und Signale  
26.03. Prozess-Lebenszyklus  
02.04. Threads und Synchr.  
09.04. Threads und Synchr.

---

23.04. IPC mit Pipes  
30.04. Sockets  
07.05. (Selbststudium)  
14.05. POSIX IPC  
21.05. (Auffahrt)  
28.05. Zeitmessung  
**04.06. Assessment**  
11.06. Terminals

Ferien

# Lernzielüberprüfung

~~Assessment I und Assessment II, beide obligatorisch.~~

~~Fliessen zu je 50% in die Gesamtbewertung ein.~~

Ein obligatorisches Assessment, 90 Minuten.

Die Schlussnote wird auf Zehntel gerundet.

Es gibt keine Modulschlussprüfung.



# Assessment

Aufgaben werden allein zu Hause am Computer gelöst.

Austeilen der Aufgaben und Abgabe mittels GitHub.

Aufgabenstellung als PDF, Lösung als TXT und C.

Alle Unterlagen\* sind erlaubt (open book).

\*Plus <http://man7.org/linux/man-pages>

Kommunikation ist nicht erlaubt.

# 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

**Alles im Assessment muss selbst geschrieben sein.**

# 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

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-fs20.slack.com/>

- |           |                                    |
|-----------|------------------------------------|
| #general  | Allg. Fragen und Ankündigungen.    |
| #random   | Eher Unwichtiges, Zufälliges.      |
| • tamberg | Messages an eine Person, "privat". |

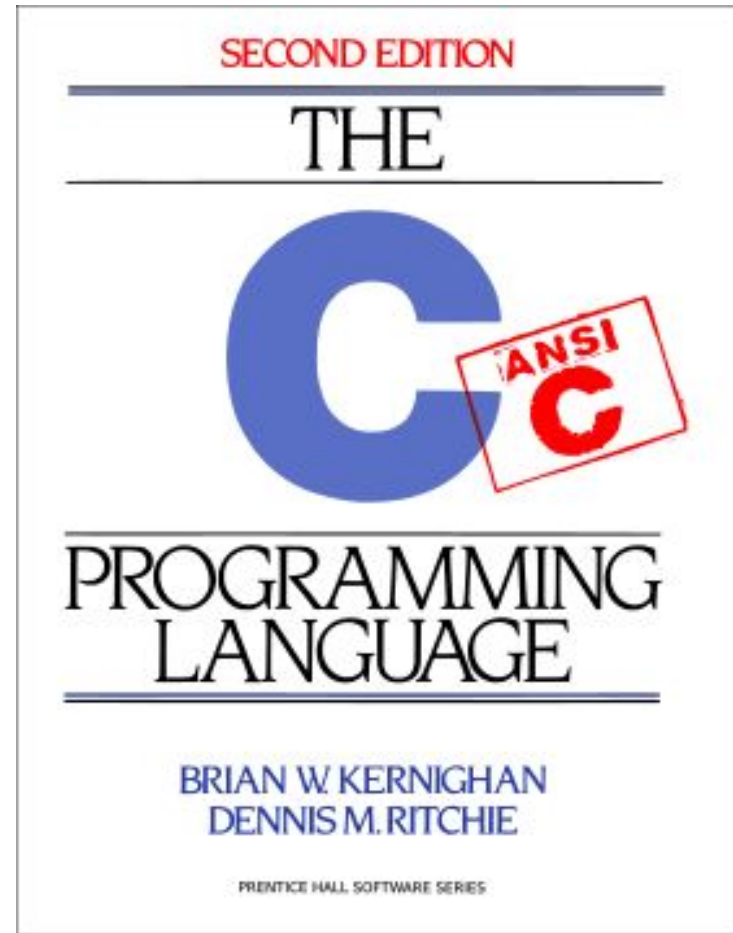
**Slack App** wird empfohlen, mobile oder Desktop.

# Literatur

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

Absoluter Klassiker für C.

270 Seiten.



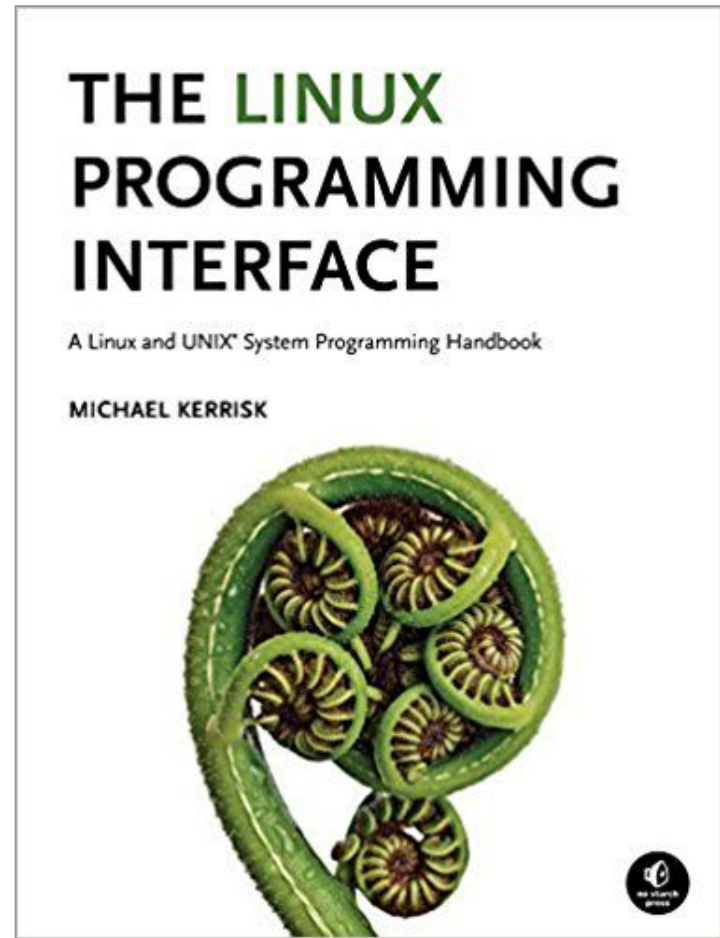


# Literatur (optional)

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

Nachschlagwerk zu  
Linux System Calls.

1500+ Seiten.

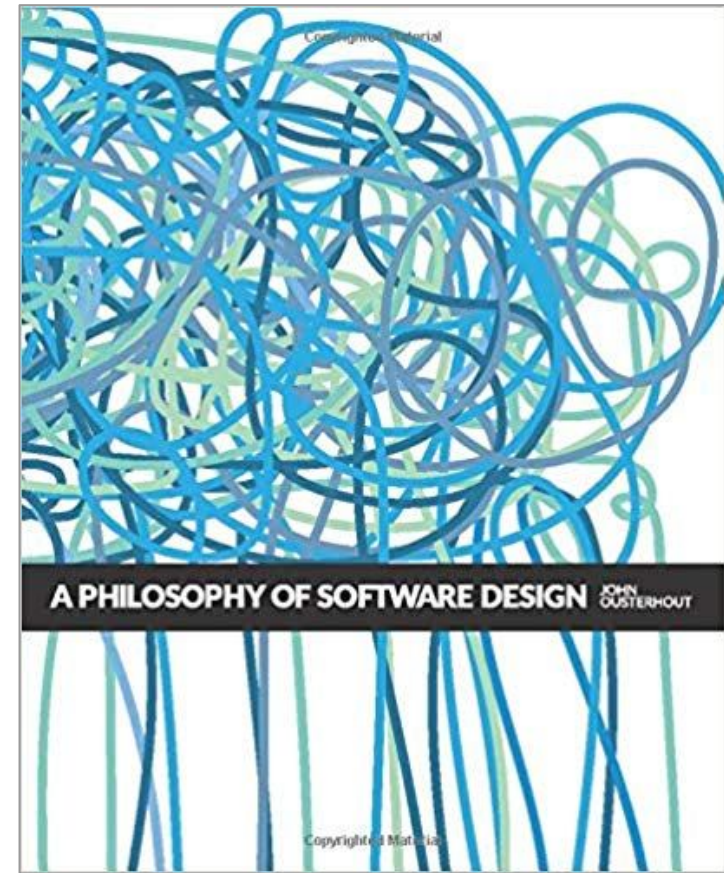


# Literatur (optional)

<https://ddg.co/?q=a+philosophy+of+software+design>

Software Engineering und  
Design von Schnittstellen.

180 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.

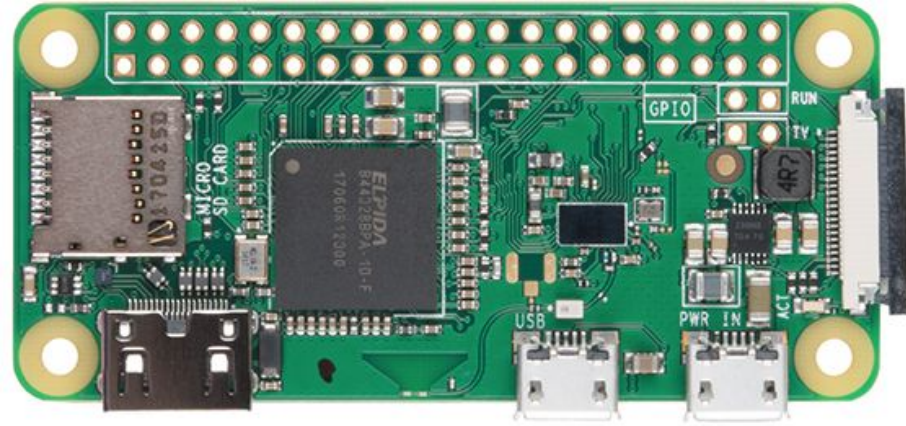
# Raspberry Pi

Einplatinencomputer:

<https://raspberrypi.org/products/raspberry-pi-zero-w/>

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

Leihweise, inklusive USB Kabel, SD Card, SD Reader.



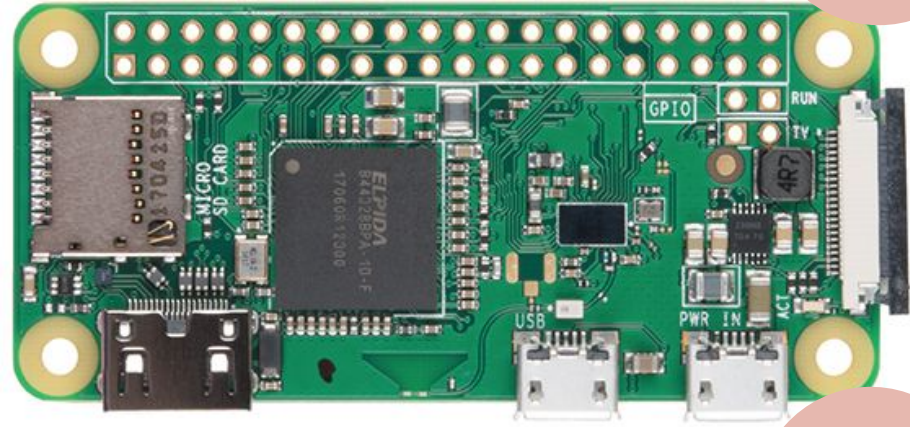
# Raspberry Pi Setup

Raspbian "Buster Lite"  
Linux IMG auf SD Card.

SD Card konfigurieren für  
Zugriff auf den Pi via USB.

SD Card in Pi einlegen, `$ ssh pi@raspberrypi.local`

Internet-Zugriff direkt mit Wi-Fi (oder via RNDIS).



# Raspberry Pi SD Card erstellen

[Etcher](#) Tool installieren, [Raspbian](#) "Buster Lite" IMG herunterladen und *mit Etcher* auf SD Card kopieren.

Fertige SD Card auswerfen, danach erneut einlegen.

Auf SD Card eine *leere* Datei namens *ssh* erstellen:

MacOS, Linux:

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

```
$ touch ssh
```

Windows:

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

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



# Raspberry Pi Zero W als RNDIS 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 Wi-Fi zu RNDIS (Mac)

SD card in Raspberry Pi einlegen.

Raspberry Pi via USB verbinden.

Auf dem MacOS Computer:

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

# Internet-Sharing Wi-Fi zu RNDIS (Win)

SD card in Raspberry Pi einlegen.

Auf dem Windows Computer:

- 1) **RNDIS Treiber installieren**
- 2) **Bonjour 3.x installieren** (~~2.x~~)
- 3) Raspberry Pi via USB verbinden
- 4) Windows Wi-Fi mit RNDIS teilen

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

# Wi-Fi Konfiguration

In Datei *wpa\_supplicant.conf* auf Pi oder SD Card:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf  
(Oder direkt auf SD Card /boot/wpa_supplicant.conf)
```

... // für Details, siehe [Raspberry Pi WiFi Doku](#)

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

# Wi-Fi Konfiguration mit EAP

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

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

```
}
```

```
    auth_alg=OPEN  
    eap=PEAP  
    identity="ORG_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
```

# Linux Shell Kommandos

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

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

# Textdatei erstellen auf Raspberry Pi/VM

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 Raspberry Pi/VM

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': Setup

*Grundlage für das  
ganze Modul syspr.*

Raspberry Pi Setup via USB zum eigenen Computer.

Oder Setup einer Linux VM auf eigenem Computer.

"Hello World" in C auf Raspberry Pi speichern.

Den C Source Code mit *gcc* kompilieren.

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

```
$ ./hello
```

# Source Code Versionierung mit Git

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

=> USER\_NAME, USER\_EMAIL

Auf dem Pi bzw. VM, *git* installieren mit *apt-get*:

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

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

User konfigurieren:

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

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

# Git konfigurieren auf Raspberry Pi/VM

## SSH Key erstellen:

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

## Raspberry Pi bzw. VM **SSH Key eintragen** auf **GitHub**:

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

# GitHub Repository klonen

GitHub Repository klonen (auf zwei Arten möglich):

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

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

Neue Datei hinzufügen:

```
$ cd REPO
```

```
$ nano my.c
```

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

# Git verwenden

Geänderte Dateien anzeigen:

```
$ git status
```

Änderungen committen:

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

Änderungen pushen:

```
$ git push
```

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

# Hands-on, 20': GitHub

*Grundlage für das  
ganze Modul syspr.*

GitHub Account einrichten, falls keiner vorhanden.

Git auf Pi bzw. VM installieren und 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

Als Vorbereitung auf die nächste Lektion, *Erste Schritte in C*, lesen Sie diese zwei Kapitel in [K&R]:

*Chapter 5: Pointers and Arrays*

*Chapter 6: Structures*

# Feedback oder Fragen?

Gerne auf <https://fhnw-syspr-fs20.slack.com/>

Oder per Email an [thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

Slides, Code & Hands-on: [tmb.gr/syspr-0](https://tmb.gr/syspr-0)

