

System-Programmierung

o: Einführung

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



Ablauf heute

⅓ Vorlesung,
⅔ Hands-on,
Feedback.

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



Hallo

Thomas Amberg (@[tamberg](https://twitter.com/tamberg)), Software Ingenieur.
FHNW seit 2018 als "Prof. für Internet of Things".
Gründer von [Yaler](https://yaler.ch), "sicherer Fernzugriff für IoT".
Organisator der [IoT Meetup](https://io.meetup.com/) Gruppe in Zürich.

thomas.amborg@fhnw.ch

3

Aufbau Modul *syspr*

15 * 3 = 45 Stunden Unterricht:

⅓ Vorlesung plus ⅔ Hands-on.

Dazu ca. 45 Stunden Selbststudium.

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

4

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.

5

Termine FS20 — Klasse 4ibb1

18.02. Einführung	21.04. IPC mit Pipes	Ferien
25.02. Erste Schritte in C	28.04. Sockets	
03.03. Funktionen	05.07. (Projektwoche)	
10.03. File In-/Output	12.05. Terminals	
17.03. Prozesse und Signale	19.05. POSIX IPC	
24.03. Prozess-Lebenszyklus	26.05. Zeitmessung	
31.03. Threads und Synchr.	02.06. Assessment II	
07.04. Assessment I	09.06. Weitere Arten von I/O	

6

Termine FS20 — Klasse 4ibb2

20.02. Einführung	23.04. IPC mit Pipes	Ferien
27.02. Erste Schritte in C	30.04. Sockets	
05.03. Funktionen	07.07. (Projektwoche)	
12.03. File In-/Output	14.05. Terminals	
19.03. Prozesse und Signale	21.05. (Auffahrt)	
26.03. Prozess-Lebenszyklus	28.05. POSIX IPC	
02.04. Threads und Synchr.	04.06. Zeitmessung	
09.04. Assessment I	11.06. Assessment II	

7

Lernzielüberprüfung

Assessment I und Assessment II, beide obligatorisch.

Fliessen zu je 50% in die Gesamtbewertung ein.

Die Schlussnote wird auf Zehntel gerundet.

Es gibt keine Modulschlussprüfung.

8

Assessments

C-Referenzkarte wird vom Dozenten verteilt.

Weitere Unterlagen sind nicht erlaubt.

Die Assessments sind schriftlich.

Die Dauer ist je 90 Minuten.

9

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.

10

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.

11

Hands-on Sessions

"[Be excellent to each other](#)", Fragen / Helfen ist OK.

Google ([DDG.co](#), ...) nutzen um Fehler zu beheben.

Blind kopieren bringt keine neuen Einsichten.

Fremden, guten Code lesen hingegen schon.

12

Ablage Slides, Code & Hands-on

<http://tmb.gr/syspr> →

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

01/
 hello.c
 README.md → Slides, Hands-on
02/
 ...

13

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.

14

Kommunikation mit Slack

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

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

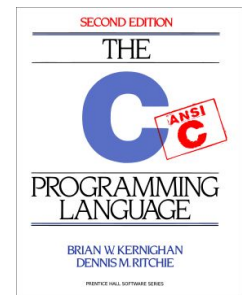
Slack App wird empfohlen, mobile oder Desktop.

15

Literatur

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

Absoluter Klassiker für C.
270 Seiten.

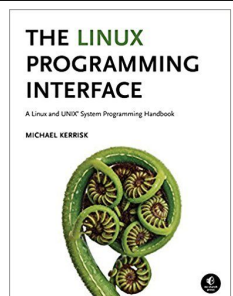


16

Literatur (optional)

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

Nachschlagwerk zu
Linux System Calls.
1500+ Seiten.

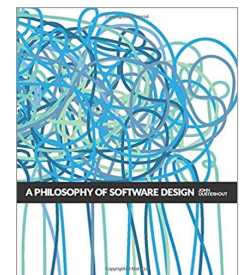


17

Literatur (optional)

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

Software Engineering und
Design von Schnittstellen.
180 Seiten.



18

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.

19

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.

20

Wieso Raspberry Pi?

Günstige Hardware.

Einheitliche Linux Plattform.

Separates System => "Sandbox".

SD Card neu schreiben => "Factory reset".

Embedded Linux Systeme sind relevant für IoT.

21

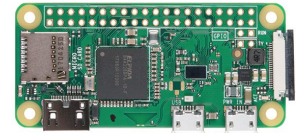
Raspberry Pi

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, gegen Unterschrift.



22

Raspberry Pi Setup

[Raspbian](#) "Buster Lite"

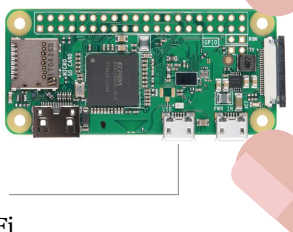
Linux IMG auf SD Card

SD Card konfigurieren für

a) Internet-Zugriff via [USB](#)

b) Internet-Zugriff via Wi-Fi

IP suchen, Login via SSH



23

Raspberry Pi SD Card erstellen

[Etcher](#) Tool installieren, [Raspbian](#) "Buster Lite" IMG
runterladen 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
```

24

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.)

25

Internet-Sharing Wi-Fi zu USB (MacOS)

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

26

Internet-Sharing Wi-Fi zu USB (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

27

Wi-Fi Konfiguration zu Hause (optional)

Auf dem Pi 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*

28

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"
}
```

29

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
```

30

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!

31

Linux Shell Kommandos

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

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

32

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
```

33

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
```

34

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
```

35

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
```

36

Source Code Versionierung mit Git

Account erstellen auf [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
```

Installation prüfen:

```
$ git
```

37

Git konfigurieren auf Raspberry Pi/VM

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
```

38

GitHub konfigurieren

Raspberry Pi bzw. VM **SSH Key eintragen** auf [GitHub](#):

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

Auf Pi bzw. VM, Passphrase ablegen in *keychain*:

```
$ sudo apt-get install keychain
$ keychain ~/.ssh/id_rsa
$ . ~/.keychain/SSHNAME-sh
(Bei Reboot wird der keychain Cache gelöscht.)
```

39

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
```

40

Git verwenden

Geänderte Dateien anzeigen:

```
$ git status
```

Änderungen committen:

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

Änderungen pushen:

```
$ git push
```

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

41

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.

42

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.

43

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

