

System-Programmierung

0: Einführung

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

Slides: tmb.gr/syspr-0

Überblick

Diese Lektion ist die *Einführung* bzw. das Drehbuch:

Was Sie vom Modul *syspr* erwarten können.

Was von Ihnen erwartet wird.

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.

Email 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 FS21 — Klasse 4ibb1

| | | | |
|--------|----------------------|--------|-----------------|
| 22.02. | Einführung | 26.04. | IPC mit Pipes |
| 01.03. | Erste Schritte in C | 03.05. | Sockets |
| 08.03. | Funktionen | 10.05. | Kein Unterricht |
| 15.03. | File In-/Output | 17.05. | POSIX IPC |
| 22.03. | Prozesse und Signale | 24.05. | Zeitmessung |
| 29.03. | Prozess-Lebenszyklus | 31.05. | Terminals |
| 05.04. | Threads und Synchr. | 07.06. | Assessment II |
| 12.04. | Assessment I | 14.06. | Abschluss |
| 19.04. | Kein Unterricht | | |

Termine FS21 — Klasse 4ibb2

| | | | |
|--------|----------------------|--------|-----------------|
| 24.02. | Einführung | 28.04. | IPC mit Pipes |
| 03.03. | Erste Schritte in C | 05.05. | Sockets |
| 10.03. | Funktionen | 12.05. | Kein Unterricht |
| 17.03. | File In-/Output | 19.05. | POSIX IPC |
| 24.03. | Prozesse und Signale | 26.05. | Kein Unterricht |
| 31.03. | Prozess-Lebenszyklus | 02.06. | Zeitmessung |
| 07.04. | Threads und Synchr. | 09.06. | Assessment II |
| 14.04. | Assessment I | 16.06. | Abschluss |
| 21.04. | Kein Unterricht | | |

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.

Assessment I und II, in Präsenz:

1 A4-Blatt* handgeschriebene Zusammenfassung.

Weitere Unterlagen (Slides, ...) sind *nicht* erlaubt.

Kommunikation (Smartphone, ...) ist *nicht* erlaubt.

Das Assessment ist schriftlich, dauert 90 Minuten.

*Beidseitig beschrieben.

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

Kommunikation via Slack

Kommunikation via Slack*, Einladung per Email:

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

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

*[Slack App](#) wird empfohlen, mobile oder Desktop.

Hybrid via Webex, Slides auf GitHub

Hybrid Unterricht via Webex, Link jeweils in Slack.

Slides und verlinkte Code-Beispiele auf GitHub:

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

Slides, Beispiele und Hands-on sind Prüfungsstoff.

Hands-on mittels GitHub Classroom

Kurze Hands-on Übungen während den Lektionen.

Private* Repos via GitHub Classroom, Link in Slack.

Jeweils Review von zwei, drei Lösungsvorschlägen.

Auch unfertige Lösungen können interessant sein.

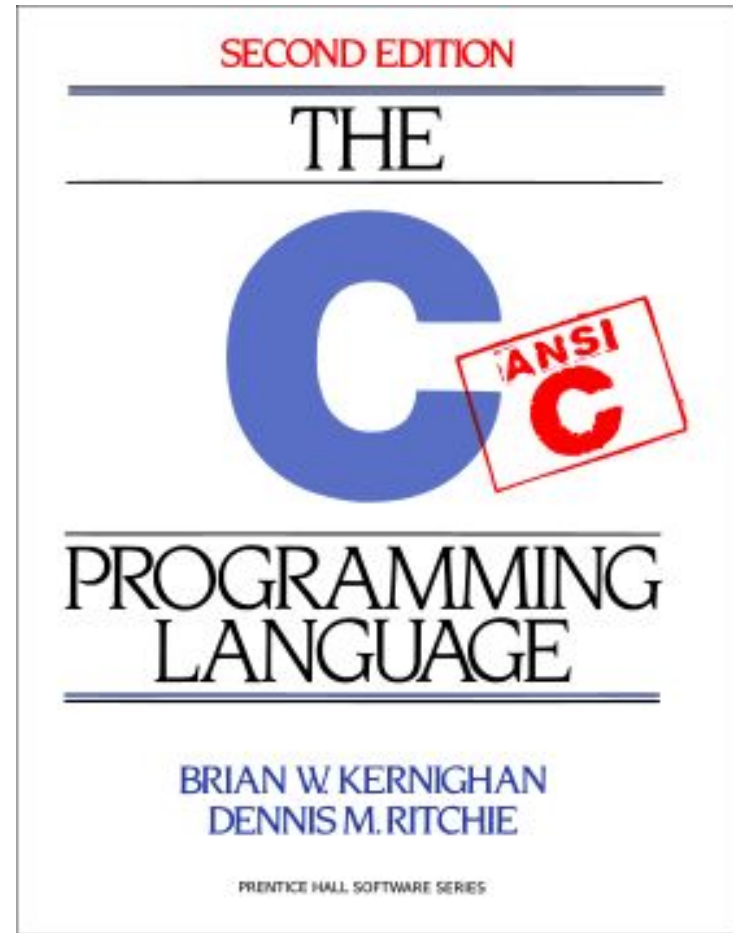
*Sie und ich sehen den Inhalt.

Literatur

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

Absoluter Klassiker für C.

270 Seiten.

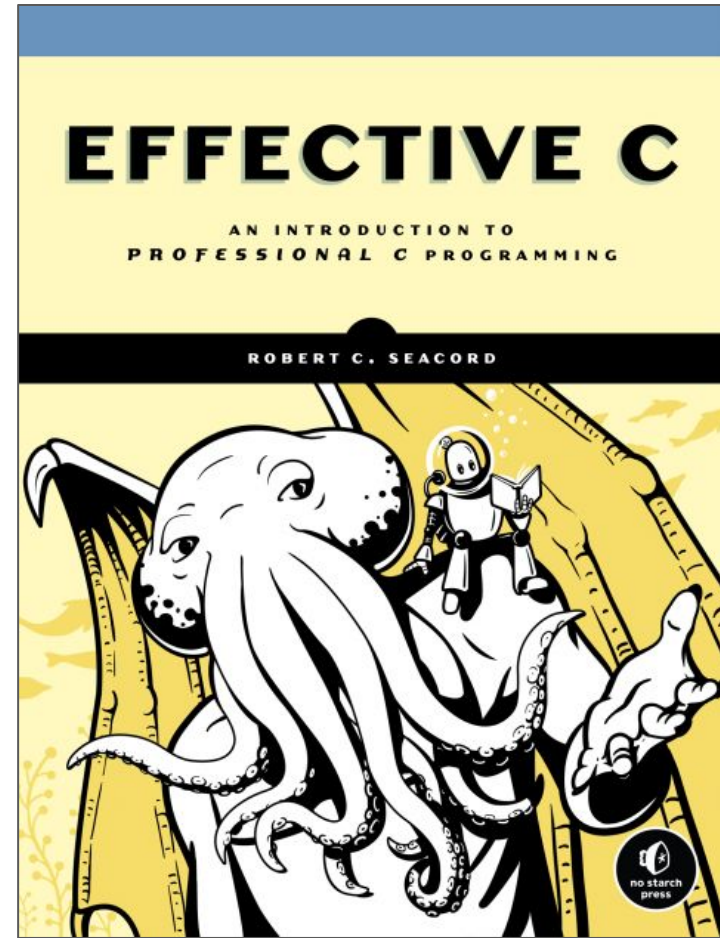


Literatur (optional)

[https://nostarch.com/
Effective_C](https://nostarch.com/Effective_C)

Sehr gute Einführung in C.

272 Seiten.

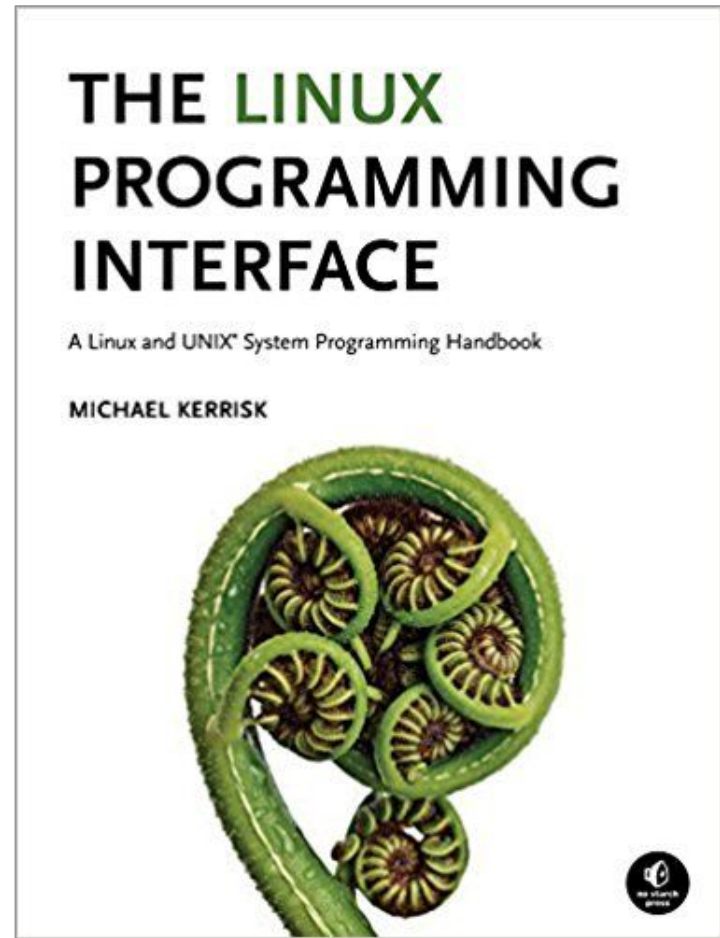


Literatur (optional)

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

Nachschlagwerk zu
Linux System Calls.

1500+ Seiten.

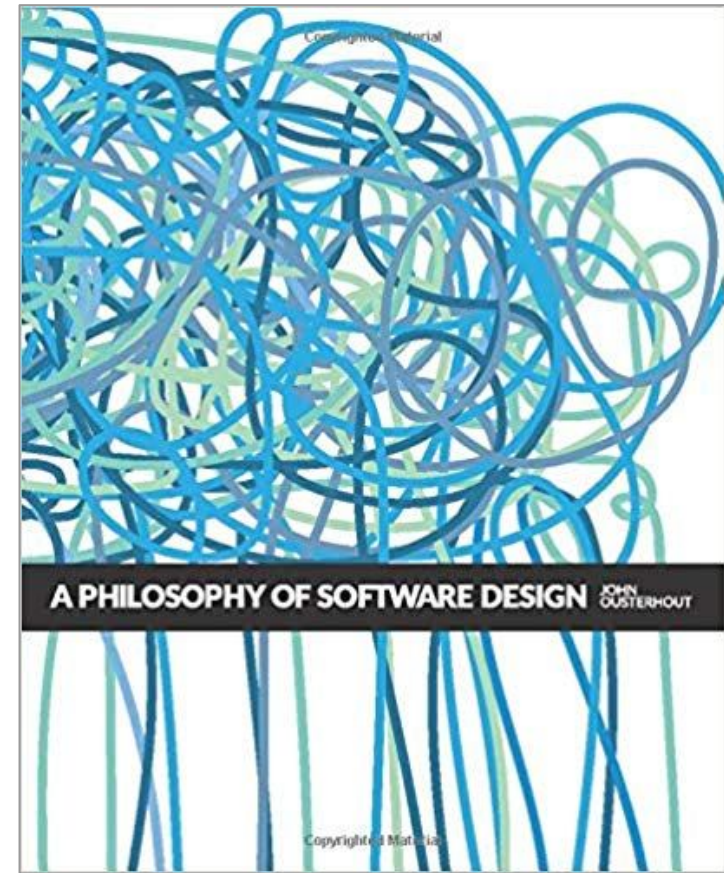


Literatur (optional)

<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* mit Flag *-std=c99*

Code Versionierung mit *git*.

Einfache Tools, ohne "Magie" => Verständnis.

Linux VM oder Raspberry Pi

System-Programmierung am Beispiel von Linux.

Die Beispiele wurden auf Raspbian entwickelt.

Im Prinzip sollte der C Code portabel sein.

Debian oder Ubuntu funktionieren gut.

WSL ist nicht empfohlen.

Wieso Raspberry Pi?

Günstige Hardware.

Einheitliche Linux Plattform.

Separates System => "Sandbox".

SD Card neu schreiben => "Factory reset".

Embedded Linux Systeme sind relevant für IoT.

Linux 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](#)).

Hands-on, 30': Setup

Setup einer Linux VM auf dem eigenem Computer.

Oder Setup eines Raspberry Pi via USB, Computer.

"Hello World" als *hello.c* auf VM bzw. 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 der VM bzw. Pi, *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"
```

SSH Keys konfigurieren (optional)

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

GitHub Account einrichten, falls nicht vorhanden.

Git auf VM bzw. Pi installieren und konfigurieren.

Dann das Hands-on Repo* auf VM oder Pi klonen.

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

*Classroom Link wird im Slack bekannt gegeben.

Zusammenfassung

Sie haben alle wichtigen Informationen zum Modul.

Sie haben eine Linux VM oder Raspbian aufgesetzt*.

Sie haben die Tools *gcc* und *git* installiert, getestet*.

Sie sind bereit für *Erste Schritte in C*.

*Wird ab der nächsten Lektion vorausgesetzt.

Feedback oder Fragen?

Gerne im Slack <https://fhnw-syspr.slack.com/>

Oder per Email an thomas.amberg@fhnw.ch

Danke für Ihre Zeit.



Tweet



The Best Linux Blog In the Unixverse

@nixcraft



Poll: Is C programming language still worth learning in 2020?

yes

79.7%

no

20.3%

10,790 votes · Final results

9:04 PM · Aug 29, 2020 · Twitter Web App

63 Retweets **18** Quote Tweets **176** Likes



Lulz4l1f3 @Lulz4l1f3 · Aug 30, 2020



Replying to @nixcraft

Sure, why not? C can be compiled into machine code, as it's one level of