

System-Programmierung

0: Einführung

CC BY-SA 4.0, T. 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.

Ist C noch relevant?

C kompiliert auf praktisch jedem Computer, egal ob Server, Desktop, mobile oder embedded System.

Glibc, Linux, OpenSSL, Curl*, Nginx, SQLite, ... sind in C geschrieben, laufen auf Milliarden von Geräten.

Neuere Sprachen haben Vorteile, aber **C hat Bestand.**

*Curl ist C auf 100 OS mit 20+ Md. Inst., inkl. Mars. 6

Wieso System-Calls?

System-Calls sind *die* Schnittstelle von Programmen im Userspace zum Betriebssystem, in *jeder* Sprache.

Um Ressourcen wie Speicher, Files oder Sockets zu nutzen, ruft ein Programm jeweils System-Calls auf.

Das Linux System-Call API ist in C dokumentiert*.

*in *man* Pages, z.B. für den System-Call `open()`.

Termine FS24 — Klasse 4ibb1

20.02. Einführung	23.04. IPC mit Pipes
27.02. Erste Schritte in C	30.04. Sockets
05.03. Funktionen	07.05. Kein Unterricht
12.03. File In-/Output	14.05. POSIX IPC
19.03. Prozesse und Signale	21.05. Zeitmessung
26.03. Prozess-Lebenszyklus	28.05. Terminals
02.04. Kein Unterricht	04.06. Assessment II
09.04. Threads und Synchr.	11.06. Abschluss
16.04. Assessment I	

Termine FS24 — Klasse 4ibb2

22.02.	Einführung	25.05.	IPC mit Pipes
29.02.	Erste Schritte in C	02.05.	Sockets
07.03.	Funktionen	09.05.	Kein Unterricht
14.03.	File In-/Output	16.05.	POSIX IPC
21.03.	Prozesse und Signale	23.05.	Zeitmessung
28.03.	Prozess-Lebenszyklus	30.05.	Terminals
04.04.	Kein Unterricht	06.06.	Assessment II
11.04.	Threads und Synchr.	13.06.	Abschluss
18.04.	Assessment I		

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.

Beispiele für [Assessment I](#) und [Assessment II](#).

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, **ohne Computer.

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 Teams

Kommunikation via Teams, Einladung per Email.

General	Ankündigungen und Fragen
C-Lang	Code, Fragen zu C Programmierung
Linux	Code, Fragen zu Linux System Calls
Random	Entfernt Relevantes, Zufälliges

Unterricht vor Ort, Slides auf GitHub

Unterricht jeweils vor Ort, aber keine Präsenzplicht.

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* Repositories via GitHub Classroom Links.

Jeweils Review von zwei, drei Lösungsvorschlägen.

Auch unfertige Lösungen können interessant sein.

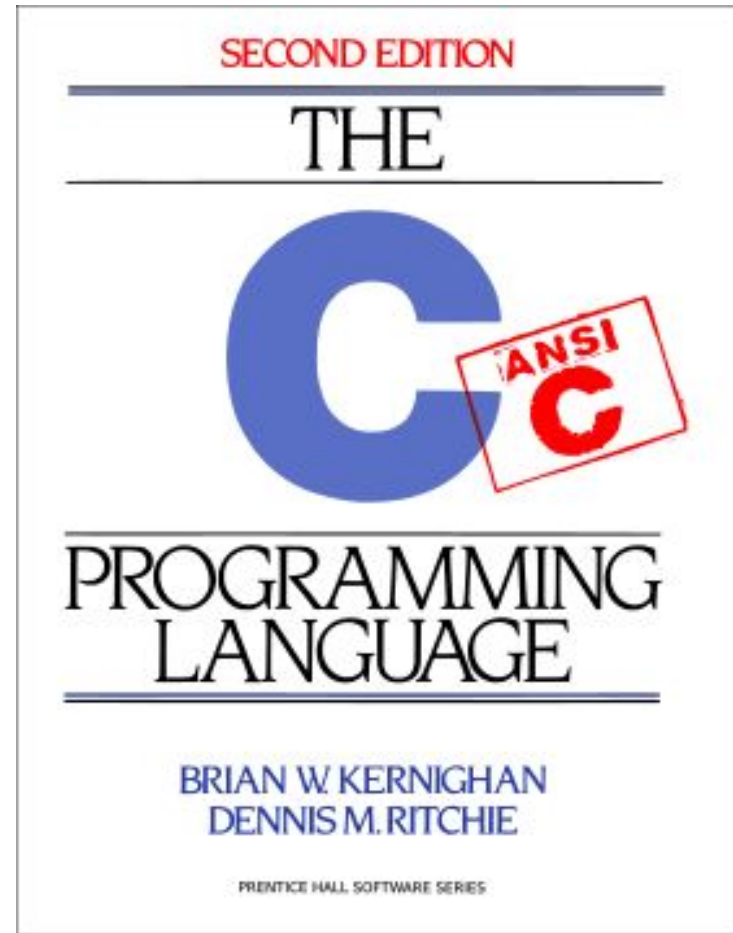
*Sie und ich sehen den Inhalt.

Literatur

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

Absoluter Klassiker für C.

270 Seiten.

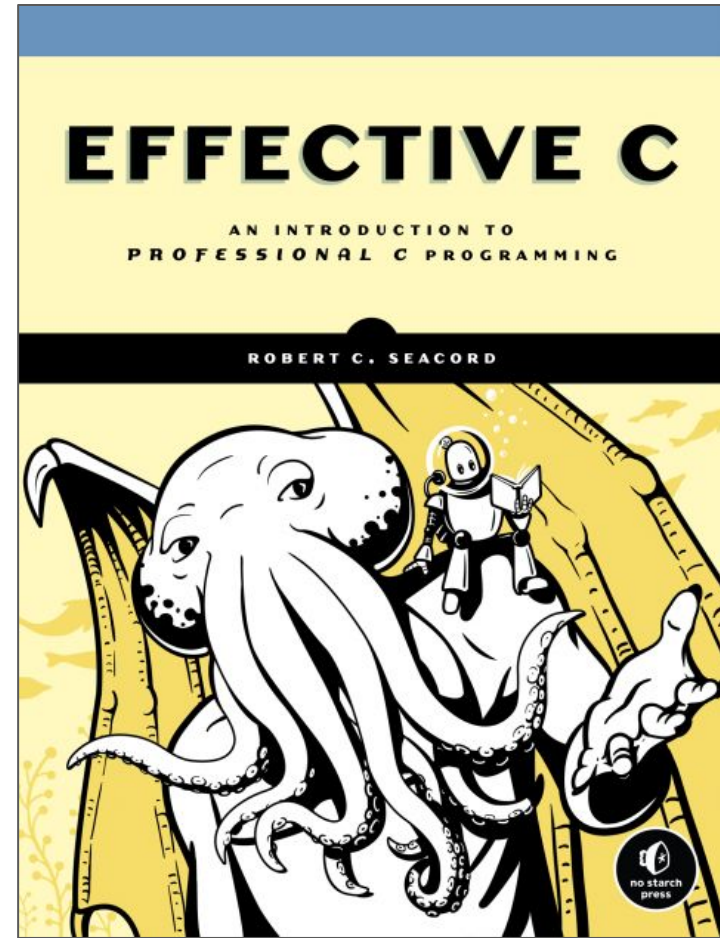


Literatur (optional)

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

Sehr gute Einführung in C.

272 Seiten.

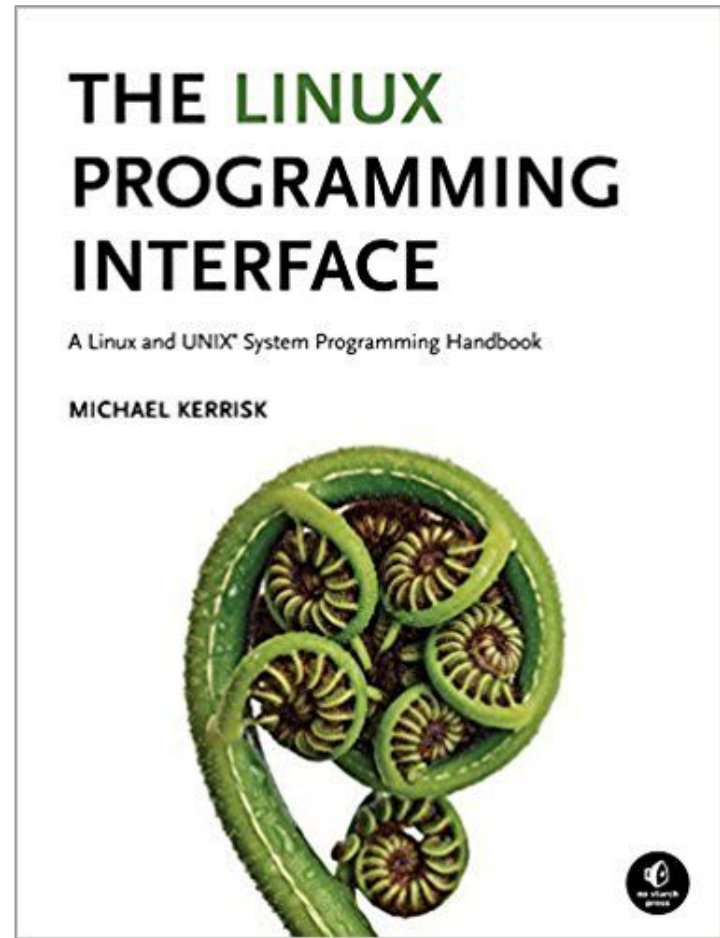


Literatur (optional)

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

Nachschlagwerk zu
Linux System Calls.

1500+ Seiten.

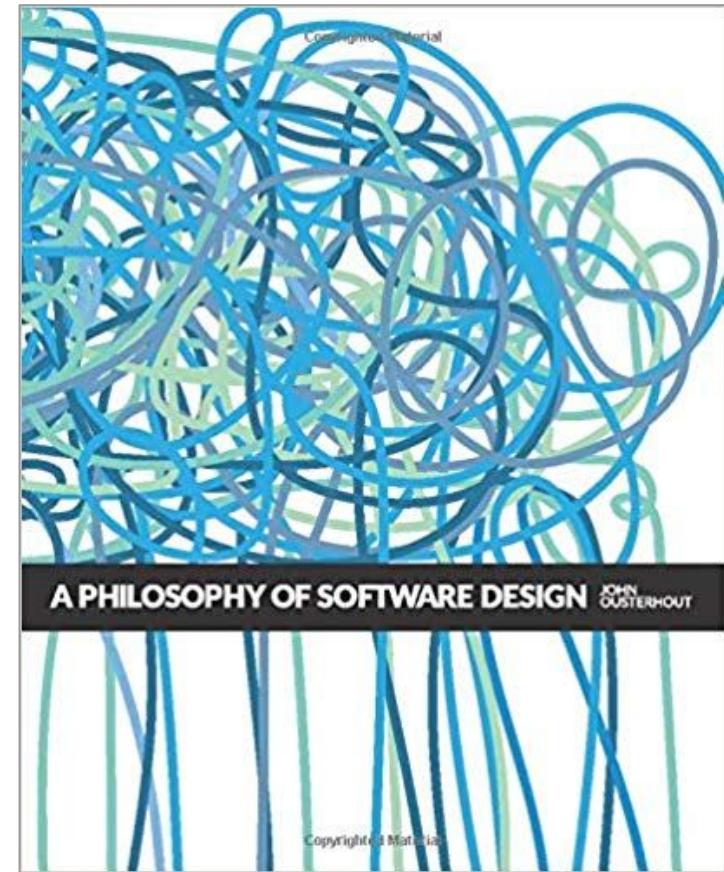


Literatur (optional)

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

Software Engineering und
Design von Schnittstellen.

180 Seiten.



Tools

Command Line bzw. Shell, via *Terminal*.

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.

MacOS native, WSL 2 oder Docker sind nicht ideal. 21

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 Teams bekannt gegeben.

Hands-on, 5': TLPI Beispiele

TLPI Beispiele ([GNU GPLv3](#)) Setup auf VM oder Pi:

```
$ wget http://man7.org/tlpi/code/download/\
tlpi-210511-book.tar.gz
```

```
$ tar xfzmv tlpi-210511-book.tar.gz
```

```
$ cd tlpi-book
```

```
$ sudo apt-get install libcap-dev
```

```
$ sudo apt-get install libacl1-dev
```

```
$ make
```

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 in Teams, 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, so it's one level of