

BETRIEBSSYSTEME - PRAKTIKUM 0

Sommersemester 2022

- Hinweise:
- Im Moodle-Kurs eine virtuelle Maschine mit der Sie die Übungen durchführen können
 - Alternativ können Sie die Übung auf einer beliebigen anderen Linux-Installation durchführen.

AUFGABE 1: KOMPILIEREN DES ERSTEN PROGRAMMS

Compiler und Linker erzeugen ein ausführbares Programm. Mit Compiler-Flags kann man das Verhalten der beiden Tools verändern, Symbole definieren und vieles mehr. Beispiel:

```
gcc -Wall -Werror -O0 -g -o test test.c
```

Mit dem vorherigen Kommando wird der Compiler angewiesen ein Programm mit dem Namen *test* aus der Quell-Datei *test.c* zu erzeugen. Der Parameter *-O0* gibt an, dass keine Optimierungen durchgeführt werden sollen. *-Wall* gibt an dass alle Warnungen erzeugt werden sollen. *-Werror* zwingt den Compiler dazu, Warnungen als Fehler zu betrachten und abzubrechen. Der Parameter *-g* gibt an, dass Debug-Informationen in das finale Programm aufgenommen werden soll.

Nutzen Sie für alle folgenden Aufgaben die Kommandos:

```
-O0 -Wall -Werror -g
```

- Laden Sie das Programm *helloWorld.c* aus dem Moodle-Kurs herunter
- Kompilieren Sie es unter Verwendung der Flags zu einem ausführbarem Programm
- Führen Sie das fertig kompilierte Programm aus

AUFGABE 2: MAKEFILES

Compiler-Befehle tippen ist aufwändig. Makefiles können uns die Aufgaben abnehmen. *make* ist ein Befehl um den Prozess zu vereinfachen. Er verwendet eine Datei mit dem Namen *Makefile* oder *makefile* um eine Kompilierung durchzuführen.

Die *Makefile* enthält die Anleitung wie ein Programm zu kompilieren ist.

Syntax:

```
target: prerequisites
    receipt
    ...
```

- *target* - Name des Outputs (z.B. Ausgabedatei)
- *prerequisites* - Inputs (z.B. Quellcode-Dateien)
- *receipt* - Was zu tun ist (jeweils ein Tab vor einer Zeile) mit Receipt

Die Kompilierung wird gestartet indem der Befehl *make* ausgeführt wird. Dies baut das erste Target in der Makefile. Mit dem Befehl *make <target_name>* können Sie ein spezifisches Target bauen.

make ist intelligent und führt Regeln nur aus, wenn es auch notwendig ist. Im nachfolgenden Beispiel wird die Kompilierung nur durchgeführt, wenn sich eine der Dateien *foo.c* oder *foo.h* verändert haben, d.h. wenn *foo.o* einen älteren Zeitstempel hat als *foo.c* oder *foo.h*:

```
foo.o: foo.c foo.h
gcc -c -o foo.o foo.c
```

Das *target* muss keine Datei sein, sondern kann ein sogenanntes *PHONY* target sein:

```
.PHONY: clean
clean:
    rm *.o temp
```

- a) Erstellen Sie ein Makefile um die das Programm *helloWorld* aus Aufgabe 1 zu kompilieren
- b) Fügen Sie ein zweites Target *clean* hinzu, welches die generierten Dateien löscht

AUFGABE 3: C-CODEBEISPIELE EVALUIEREN

- a) Evaluieren Sie Codebeispiele aus den Vorlesung-Abschnitt "Praktikum - Linux und C-Programmierung"

AUFGABE 4: FEHLER IN C-PRPROGRAMM BEHEBEN

Laden Sie sich das Programm *fix_me.c* aus dem Moodle-Kurs. Das Programm versucht einen String in der Variable *src* nach *dst* und *new* zu kopieren.

Es enthält allerdings einige Fehler.

- a) Schreiben Sie eine Makefile, die die Datei *fix_me.c* in ein ausführbares Programm kompiliert. (Achtung: Das Programm wirft zu viele Fehler während des Compile-Vorgangs).
- b) Ändern Sie die Datei *fix_me.c* so ab, dass das Programm korrekt kompiliert.
- c) Führen Sie das Programm aus. Stürzt es ab?
- d) Das Programm *cppcheck* untersucht Quellcode nach Fehlern. Das Programm *valgrind* untersucht Programme während der Ausführung nach Speicherproblemen. Nutzen Sie beide Programme um das Programm zu korrigieren.

Syntax für *cppcheck*:

```
cppcheck <c-source-file>
```

Syntax für *valgrind*:

```
valgrind <ausfuehrbare datei>
```

Referenzen

make Dokumentation: <https://www.gnu.org/software/make/manual/>

cppcheck Dokumentation: <http://cppcheck.sourceforge.net/manual.pdf>

valgrind Dokumentation: <https://www.valgrind.org/docs/manual/manual.html>