BS I WS 2019/20: Wiederholung Makefiles

Eric Ackermann, Felix Gohla, Otto Kißig
bs@hpi.uni-potsdam.de
Freenode-IRC: #hpi-bs1-2019 @EightSQ, @xDEAD10CC

Hasso-Plattner-Institut — 11. November 2019

1 Wie C-Programme zum Leben erweckt werden...

Programme, die wir in C schreiben werden vom Compiler in ausführbare Dateien (auch *executables* oder *binaries* genannt) übersetzt. Die Datei, in die wir unseren Quelltext schreiben (üblicherweise mit der Endung .c), ist also nicht die, die am Ende auch ausgeführt wird. Der wohl sichtbarste Unterschied zwischen beiden: unsere Quelldatei ist Klartext, und wir können sie in einem Editor öffnen und lesen, während das *binary* fast ausschließlich aus unleserlichen Binärdaten besteht.

Zu diesem Prozess der Übersetzung in Binärdateien werden einige Schritte durchlaufen. Wenn sie nun ein eigenes kleines C-Programm übersetzen, dann geschieht das ungefähr so:

foo.c

```c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Hello, Foo!\n");
    return 0;
}
```

Command Line

```
$ gcc -o foo foo.c
$ ./foo
Hello, Foo!
```

Der GCC[1] führt dabei mehrere Teilschritte aus, da Sie lediglich eine C-Datei eingeben und eine Ausgabedatei angeben. Im Hintergrund wird der Compiler jedoch logischerweise Ihren Quelltext einlesen, prozessieren und kompilieren. Dabei entsteht ein sogenanntes *Object-File*[2], welches zwar den in Maschinensprache übersetzten Code Ihrer übersetzten Datei enthält, selbst als solches aber noch nicht ausführbar ist. Dazu benötigt es einen *Linker*, der mehrere Object-Files und eventuelle Libraries zu einer ausführbaren Datei zusammenbindet.

Sie können sich vorstellen, dass in größeren Projekten keinesfalls der gesamte Quelltext in eine einzelne Datei geschrieben wird, aus der dann mit einem solchen Aufruf des GCC eine ausführbare Datei erzeugt wird. Nehmen wir also an, Ihr Projekt hat nun gut 42 verschiedene Quelltext- und Headerdateien, in denen Sie den Code für Ihre Applikation dekomponiert haben. Es wäre jetzt sehr umständlich, nachdem Sie zum hundertsten Mal ein Semikolon vergessen haben, jedes Mal den Compileraufruf mit 42 Eingabedateien in Ihre Kommandozeile einzutippen.

[1] C-Compiler der GNU Compiler Collection
[2] Mit dem Flag -c können Sie dem Compiler mitteilen, dass sie die Datei nur kompilieren, nicht aber binden wollen.

2 Aufgaben

2.1 Recherche

Sehen Sie sich die Einführung zum MAKE Buildsystem von Sven Köhler an: Vorlesung PT I WS 2017/2018. Machen Sie sich unter anderem mit den Begriffen **Makefile**, **Targets** und **Regel** vertraut. Weitere Informationen und Ressourcen werden Ihnen im GitLab auf der Seite Buildsysteme zur Verfügung gestellt.

Beantworten Sie die folgenden Fragen:

(a) Woran erkennt MAKE, wann welche der angegebenen Regeln für welche Dateien ausgeführt werden müssen?

(b) Was bedeutet in diesem Zusammenhang der Begriff "Pattern-Rules"?

(c) Was sind sogenannte `.PHONY` Targets?

(d) Was ist das DEFAULT Target?

2.2 Programmierung

Schreckliches ist passiert! Das Rasenmäherschaf wurde gehackt und Bösewichte haben die interne Schaf-Datenbank gelöscht. Nun weiß es gar nicht mehr, wann es wo mä(ää)hen soll. Glücklicherweise konnte eine mysteriöse Datei mit einer Notiz gefunden, die wir Ihnen weiterleiten.
Die Notiz bestand aus den Worten **"C11"**.

Folgende Dateistruktur ist vorhanden:

Command Line

```
$ tree
.
|-- Makefile
|-- encrypted_coordinates.db
`-- src
    |-- caesar.c
    |-- caesar.h
    |-- read_sheep_db.c
    |-- sheep_coordinate.c
    `-- sheep_coordinate.h
```

(a) Vervollständigen Sie die Datei `src/caesar.c`, um die Verschlüsselung der Koordinatendatenbank zu knacken.

(b) Schreiben Sie nun noch ein `Makefile`, das die Quelldateien in das Programm `read_sheep_db` übersetzt. Es soll unter anderem die Targets ALL und CLEAN anbieten zum Bauen, bzw. Löschen des Programms (ausführbare Datei).

(c) Dekodieren Sie die Datenbank mithilfe des Programms und helfen Sie dem Schaf, endlich wieder fahren zu können. Speichern Sie die entschlüsselte Datei als `sheep.db` und geben Sie diese mit ab.

Bei der Abgabe zu beachten: Geben Sie alle Textaufgaben in einer Textdatei `series-1.txt` und all den von Ihnen geschriebenen Programmcode (zusammen mit der Textdatei) im Wurzelverzeichnis eines gzip-komprimierten Tarfile `series-1.tar.gz`.
Achten Sie darauf, dass keine Kompilate, temporäre Dateien Ihres Editors oder versteckte Dateien (etwa ein `.git`-Ordner) enthalten sind.