Makefiles Introduction

Felix Graner
3. Mai 2023

1 Makefiles

Wie man schon nach einigem Herumprobieren und Debuggen von Assembler- oder auch generell Code feststellt, ist dass es doch unangenehm Arbeitsaufwändig ist die Assemblier- und Linkbefehle jedes mal neu zu schreiben oder aus dem Eingabeverlauf zu suchen. Um hier einen einfachen und angenehmen Weg vorzustellen mit dem dies unnötig wird wollen wir kurz Makefiles vorstellen.

Makefiles sind eigentlich dazu gedacht das Assemblieren und Linken von größeren Programmen zu vereinfachen, so dass man auch problemlos Unterprogramme auslagern kann und nicht immer wieder alles händisch einbinden muss wenn man ein Detail ändert. Zusätzlich werden Dateien auch automatisch nur neu Compiliert wenn es wegen Abhängigkeiten nötig wird. Sie sind aber praktisch immer nützlich sobald man mehr als einen Befehl zum vollständigen Compilieren benötigt. Zusätzlich kann man auch recht einfach seinen Ordner automatisiert aufräumen, dazu aber später mehr.

1.1 How to make a Makefile

1. Gehen Sie in den Ordner in dem ihre zu compilierenden Dateien liegen
2. erstellen Sie mit dem Editor Ihrer Wahl eine Datei namens "Makefile"
3. schreiben Sie in die erste Zeile 'all: FirstProgram"
4. darunter "FirstProgram:"
5. schreiben Sie in den nächsten Zeilen jeweils die Assemblierbefehle wie Sie sie auch in die Kommandozeile eingegeben hätten
6. speichern Sie ihre Änderungen und compilieren Sie in Zukunft einfach indem Sie $make$ in die Kommandozeile eingeben

Das sollte dann etwa so aussehen:

```
all: FirstProgram

FirstProgram:
        arm-linux-gnueabihf-as -o objectfile.o programfile.s
        arm-linux-gnueabihf-gcc -o executable objectfile.o
```

Was praktisch passiert ist, dass nun von oben nach unten zeilenweise die Befehle ausgeführt werden die unter dem Sprunglabel stehen. Bei uns sind das also gerade die Befehle die benötigt werden um eine Datei namens $programfile.s$ erst in das Objectfile namens $objectfile.o$ zu übersetzten und dann vom $objectfile.o$ zu einer ausführbaren Datei namens $executable$ zu compilieren. Diese kann man danach mit ./executable starten.
Dabei kann man natürlich alle diese Namen ändern. Wichtig dabei ist nur, dass der name des objectfiles in der ersten und zweiten Zeile übereinstimmt.
Wichtig ist auch, dass man sich im klaren ist, dass die Dateien die möglicherweise vorher mit diesen Namen (objectfile.o und executable) vorhanden waren, fraglos überschrieben werden.

1.2 Benutzung von Makefiles für andere Zwecke

Da wir nun verstanden haben wie man Makefiles für das einfache Compilieren von Dateien benutzt kommt hier der versprochene Teil über das automatisierte Aufräumen:
Da unter dem Jumplabel die Zeilen ausgeführt werden, egal was dort steht kann man sich dies einfach zunutze machen. Wenn man für verschiedene Studienfächer verschiedene Ordner nutzt, fällt möglicherweise auf, dass die ”wichtigen” Dateien, also diejenigen mit Quelltext meist in der gleichen Programmiersprache geschrieben sind, somit auch die gleiche Endung haben. Das ist genau was wir uns zunutze machen werden.
Die Anfangsidee ist eine Kopie des Makefiles mit der gleichen Endung zu erstellen, dann einfach alle Dateien in dem Ordner zu löschen die eine andere Endung haben und am Ende das Makefile wiederherzustellen. Dafür benötigen wir die Befehle copy und remove, also cp und rm:

1. cp Makefile Makefile.ending
2. `rm *[^.ending]`
3. cp Makefile.ending Makefile
4. rm Makefile.ending
5. ls -l

Als Beispiel hier mit der Endung .c:

```
all: CleanUp

FirstProgram:
        arm-linux-gnueabihf-as -o objectfile.o programfile.s
        arm-linux-gnueabihf-gcc -o executable objectfile.o
        ./executable

CleanUp:
        cp Makefile Makefile.c
        rm *[^.c]
        cp Makefile.c Makefile
        rm Makefile.c
        ls -l
```

Dies funktioniert nur einwandfrei solange sich keine Unterordner in dem Ordner befinden in dem man das Makefile anwendet, da rm standardmäßig keine Ordner löschen darf und bei dem Versuch abbricht, wodurch die darauf folgenden Befehle nicht mehr ausgeführt werden. Man $kann$ dies umgehen indem man beispielsweise das -r Flag setzt. **Dies ist nicht zu empfehlen!** weil es das Unterverzeichnis mit allen enthaltenen Daten ohne Nachfrage löscht. Man sollte mindestens noch das -i Flag setzten um zu entscheiden ob eine Datei gelöscht werden soll. Als Befehl also

```
rm -r -i [^.ending]
```

In kurz also: benutzen sie diese Vorgehensweise nur in Ordnern ohne Unterordner in denen sie auch keine anderen wichtigen Dokumente aufbewahren. Falls sie aber wie in Rechnerorganisation in mindestens zwei Sprachen programmieren werden, wäre es doch nützlich Dateien mit verschiedenen Endungen erhalten zu können. Und tatsächlich ist dies eine Leichtigkeit. Man muss nur die Endungen in den eckigen Klammern anhängen. Beispielsweise

```
rm -r -i [^.c.s]
```

Trotz einigen Test unsererseits empfehlen wir dies erst einmal in einem Testordner zu testen um nicht ausversehen ihre nächten Abgaben zu löschen.

2 zum Weiterlernen

Ein ausführlicheres Tutorial gibt es hier (click) und an verschiedenen Stellen im Internet, Stichwort ”Makefile Tutorial” .