Software-Entwicklung mit *make*

Praktikum Software-Technik

Dipl.-Ing. T. Fründ

Email: fruend@et.FH-Osnabrueck.de

23. Juni 1998

1 Was leistet make

Das Programm make erleichert die Entwicklung von Programmen, die aus einer Anzahl verschiedener Source- und Headerdateien bestehen. Es erlaubt, bei Änderungen in einer Datei, automatisch die davon betroffenen Programmteile neu zu kompilieren und zu binden. Dazu generiert es anhand einer Beschreibungsdatei (meist mit Namen Makefile oder makefile) automatisch die notwendigen Kommandos und führt diese aus.

In dem Verzeichnis /home6/public/fruend/swt/script/make befinden sich die Beispieldateien[1], die in diesem Kapitel benutzt werden. So auch das folgende Makefile:

```
#
# Makefile Softwaretechnik
#
# tf.c          Testprogramm fuer die Fourieranalyse
# fourier.c     Unterprogramm-Modul fuer die Fourieranalyse
# fourier.h     Zugehoerige Headerdatei
# rmath.c       Unterprogramm-Modul zur Erweiterung der math. Funktionen
# rmath.h       Zugehoerige Headerdatei
# rfkt.c        Unterprogramm-Modul mit zu untersuchenden Fuktionen
#
# Die Headerdateien sind in /home6/public/fruend/swt/include vorgegeben
# Ebenso sind die benoetigten GKS-Pfade und -Bibliotheken als Makros definiert.
#

HEADERS = rmath.h fourier.h
SRC     = rmath.c fourier.c rfkt.c tf.c
OBJS    = rmath.o fourier.o rfkt.o tf.o

INCLUDE = -I/home6/public/fruend/swt/include
GKSINC  = -I/opt/gtsgral/gralcgks/inc
LIBS    = -lgksconv -lgralcgks -lgralcgi -lcgiclient \
          -lXol -lXt -lXmu -lX11 -lw -lintl -ldl -lm
LIBDIRS = -L/opt/gtsgral/libs -L/usr/openwin/lib \
          -L/home6/public/fruend/gdv/lib


tf:        ${OBJS}
        cc -g -o tf ${OBJS} ${LIBDIRS} ${LIBS}

tf.o:      tf.c
        cc -c -g tf.c ${INCLUDE}

fourier.o: fourier.c
        cc -c -g fourier.c ${INCLUDE} ${GKSINC}

rmath.o:   rmath.c
        cc -c -g rmath.c ${INCLUDE}

rfkt.o:    rfkt.c
        cc -c -g rfkt.c ${INCLUDE}
```

[1] In den Beispielen werden ähnliche Dateinamen wie in den Übungen benutzt. Dies dient nur zur Veranschaulichung, die Dateien selbst haben mit den Übungen **nichts** gemeinsam!

```
# Erzeugen einer Funktionsbibliothek
libfourier.a: rmath.o fourier.o
        ar r libfourier.a rmath.o fourier.o

# Erzeugen des Testprogramms unter Verwendung der neuen Library
tf_lib:    libfourier.a rfkt.o tf.o
        cc -g -o tf_lib tf.o rfkt.o -L. ${LIBDIRS} -lfourier ${LIBS}

# Mit 'make clean' das Verzeichnis aufraeumen
clean:
        /usr/bin/rm -f *.o *%

# Abhaengigkeiten erzeugen
depend:
        makedepend -- ${INCLUDE} ${GKSINC} -- ${SRC}

# DO NOT DELETE THIS LINE -- make depend depends on it.

rmath.o: /home6/public/fruend/swt/include/rmath.h /usr/include/math.h
fourier.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
fourier.o: /usr/include/stdlib.h /home6/public/fruend/swt/include/gksconv.h
...
```

2 Der Aufruf von make

Der Befehl zum Aufruf von make lautet:[2]

```
make [Optionen] Ziel
```

Beispiel:

```
make -n rmath.o
```

Das Programm make liest im aktuellen Verzeichnis die Datei Makefile und sucht dort nach einem Eintrag aus dem hervorgeht, wie zu dem angegebenen Ziel (hier rmath.o) gelangt werden kann. Die Datei Makefile ist eine ASCII-Textdatei, die vom Benutzer (z.B. mit textedit oder emacs) erstellt wird. Zeilen mit einem #-Zeichen am Anfang sind Kommentarzeilen. Ist das letzte Zeichen einer Zeile ein '\' so wird diese in der nächsten Zeile fortgesetzt.

Die Option -n veranlaßt make, die Befehle zum Erreichen des Ziels rmath.o nur zu generieren und anzuzeigen, aber nicht direkt auszuführen. Daher ist diese Option besonders nützlich zum Testen von Makefiles.

Mit der Option -f teilt man dem Kommando make den Namen der benötigten Beschreibungsdatei mit, wenn nicht die Namen Makefile oder makefile benutzt werden sollen.

[2] Für die genaue Syntax siehe man-pages bzw. Unix-Literatur

3 Das Ziel des Makefiles

Herzstück eines jeden Makefile sind Einträge der folgenden Form:

```
Ziel:    Abhaengigkeit(en)
<TAB>  Regel
```

Beispiel:

```
rmath.o:      rmath.c
        cc -c rmath.c -I/home6/public/fruend/swt
```

Die drei Teile eines solchen Eintrages können kurz als **Was**, **Warum** und **Wie** bezeichnet werden:

Ziel: Was von make neu angefertigt werden soll. In diesem Fall die Datei rmath.o (auch neudeutsch als Target bezeichnet).

Abhängigkeit(en): Warum rmath.o neu angefertigt werden muß. Falls sich also an der Datei rmath.c etwas geändert hat, geht make davon aus, daß das Ziel rmath.o neu zu erzeugen ist. Dazu vergleicht make einfach das Erstellungsdatum und -uhrzeit von rmath.o mit dem jeder einzelnen Datei aus den Abhängigkeiten (neudeutsch: Dependencies).

Regel: Wie aus den Abhängigkeiten das Ziel erzeugt werden kann (auch Rule genannt). In diesem Fall also durch Aufruf des cc Compilers für die Datei rmath.c.
Achtung: Es ist unabdingbar, daß diese Zeile (und ggf. jede Folgezeile!) tatsächlich mit einem Tabulator-Zeichen <TAB> anfängt.

Jede Abhängigkeit ist für make selbst wieder ein mögliches Ziel. Daher wird makefile so lange ausgewertet, bis alle Abhängigkeiten erkannt sind. Ein weiterer möglicher Eintrag wäre z.B.:

```
tf:     tf.o fourier.o rmath.o
      cc -o tf fourier.o rmath.o -lm -lgks
```

Jetzt wird mit dem Aufruf `make tf` das Programm tf aus den Objektdateien tf.o, fourier.o und rmath.o neu zusammengebunden, falls mindestens eine dieser Dateien jüngeren Datums als tf ist. Sollte auch z.B. fourier.o älter als eine seiner Abhängigkeiten sein, würde es zuerst neu erzeugt.

Es werden also immer die Tagrets neu erzeugt, an deren abhängigen Dateien Änderungen aufgetreten sind. Alle anderen Targets bleiben davon unberührt. Bei einem Programm bedeutet dies, daß nur die Programm-Module neu compiliert werden, deren Quelldateien sich geändert haben.

4 Die voreingestellten Regeln

Um nicht für jede Datei einen eigenen Eintrag im Makefile vornehmen zu müssen, besitzt make zahlreiche vorgegebene Regeln, wie von einer Datei zu einer anderen gelangt werden kann. Dabei orientiert sich make an den Endungen (Suffix) der Dateien. Die Voreinstellungen können auch im Makefile selbst durch Einträge der folgenden Art umgesetzt werden:

```
.c.o:
        cc -c $<
```

Hiermit wird eine allgemeine Regel[3] angegeben, wie ein Ziel mit Endung '.o' aus einer Datei mit gleichem Namenspräfix und der Endung '.c' hergestellt wird.

Diese Regel wird nur angewandt, falls es keinen expliziten Eintrag für das gewünschte Ziel gibt. Ausdrücke wie '$<' haben für make eine besondere Bedeutung, in diesem Fall der Präfix des Ziels plus die Endung der Abhängigkeit[4].

Da eine ganze Reihe von vordefinierten sog. Suffix-Regeln existieren, kann es leicht zu Konflikten mit den Voreinstellungen kommen. Daher wird häufig die Liste der von make erkannten Endungen zuerst gelöscht und dann neu definiert. Dies geschieht durch Definition des voreingestellten Ziels .SUFFIXES wobei die von make zu erkennenden Endungen danach eingegeben werden und keine Regel angegeben wird:

```
.SUFFIXES:
.SUFFIXES: .c .o
```

Damit würde make nur noch die zwei angegebenen Endungen berücksichtigen.

In einem Makefile sind zwei Regeln definiert, die automatisch aus einer '.c' bzw '.cc' Datei ein ausführbares Programm erzeugen. Das geht natürlich nur, wenn keine weiteren Objekt-Dateien dazugebunden werden müssen. Nützlich ist das, wenn ein kleines Testprogramm[5] schnell kompiliert werden soll. Dann muß keinerlei Eintrag oder Änderung in Makefile durchgeführt werden, sondern es kann direkt make <Programmname_ohne_Endung> aufgerufen werden. Den Rest macht make dann alleine.

5 Makro-Definitionen

Das letzte wesentliche Element eines Makefiles sind Makro-Definitionen, die als Textplatzhalter fungieren:

```
HEADERS = rmath.h fourier.h
SRC     = rmath.c fourier.c rfkt.c tf.c
OBJS    = rmath.o fourier.o rfkt.o tf.o

INCLUDE = -I/home6/public/fruend/swt/include
GKSINC  = -I/opt/gtsgral/gralcgks/inc
LIBS    = -lgksconv -lgralcgks -lgralcgi -lcgiclient \
          -lXol -lXt -lXmu -lX11 -lw -lintl -ldl -lm
LIBDIRS = -L/opt/gtsgral/libs -L/usr/openwin/lib \
          -L/home6/public/fruend/gdv/lib
```

Die Dereferenzierung eines Makros erfolgt mittels $(Makroname) oder ${Makroname}, im obigen Beispiel also:

```
tf:         ${OBJS}
      cc -g -o tf ${OBJS} ${LIBDIRS} ${LIBS}
```

[3] <TAB> nicht vergessen!

[4] Genauere Erklärungen zu diesen dynamischen Makros finden sich wieder mit 'man make'

[5] Testprogramme dürfen nie den Namen test bekommen. Dies führt zu Konflikten mit dem in der Shell eingebauten Kommando test.

6 Abhängigkeiten finden mit makedepend

Häufig werden von Header-Dateien weitere Header-Dateien eingebunden. Dadurch kann es zu komplexen Abhängigkeiten kommen. Das Programm makedepend dient dazu, von einer oder mehreren C/C++-Quelldateien automatisch die Abhängigkeiten zu erkennen und entsprechende Einträge im Makefile vorzunehmen und es somit anzupassen. Häufig wird der Aufruf von makedepend wieder selbst über make vorgenommen[6].

Dazu dient der Eintrag:

```
depend:
        makedepend -- ${INCLUDE} ${GKSINC} -- ${SRC}
```

Weiter unten im Makefile muß sich dann die Zeile

```
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

befinden. Der Bereich hinter dieser Zeile wird durch makedepend verwaltet und kann daher nicht für eigene Einträge verwendet werden. Das Kommando make depend veranlaßt make das Programm makedepend aufzurufen. makedepend durchsucht daraufhin die Source-Dateien (rmath.c, fourier.c und tf.c) und fügt für jede gefundene #include ... Direktive eine entsprechende Zeile in makefile hinzu und zwar hinter der oben angegebenen Kommentarzeile. Dabei ist zu beachten, daß das Makefile verändert wird[7]. Die ursprüngliche Datei wird in Makefile.bak umbenannt. In den Makros INCLUDE und GKSINC sind dabei die nach Header-Dateien zu durchsuchenden Verzeichnisse angegeben. Im obigen Beispiel werden folgende Zeilen hinzugefügt:

```
rmath.o: /home6/public/fruend/swt/include/rmath.h /usr/include/math.h
fourier.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
fourier.o: /usr/include/stdlib.h /home6/public/fruend/swt/include/gksconv.h
fourier.o: /opt/gtsgral/gralcgks/inc/gks.h /opt/gtsgral/gralcgks/inc/gksmap.h
fourier.o: /opt/gtsgral/gralcgks/inc/gkstypes.h
fourier.o: /opt/gtsgral/gralcgks/inc/gksfnams.h
fourier.o: /opt/gtsgral/gralcgks/inc/gksnames.h
fourier.o: /opt/gtsgral/gralcgks/inc/gkslnams.h
fourier.o: /opt/gtsgral/gralcgks/inc/gkserror.h
fourier.o: /home6/public/fruend/swt/include/achsen.h /usr/include/string.h
fourier.o: /home6/public/fruend/swt/include/rmath.h /usr/include/math.h
fourier.o: /home6/public/fruend/swt/include/fourier.h
rfkt.o: /home6/public/fruend/swt/include/rmath.h /usr/include/math.h
tf.o: /usr/include/stdio.h /usr/include/sys/feature_tests.h
tf.o: /usr/include/stdlib.h /usr/include/string.h
tf.o: /home6/public/fruend/swt/include/rmath.h /usr/include/math.h
tf.o: /home6/public/fruend/swt/include/fourier.h
```

Im Ablauf von makedepend sind zur Erhöhung der Laufzeit-Effizienz einige Annahmen gemacht, die dazu führen können, daß sich das Programm nicht so verhält wie vermutet. Eine wesentliche implizite Voraussetzung besteht in der Annahme, daß alle im Aufruf von makedepend angegebenen Quell-Dateien im wesentlichen die gleichen Header-Dateien einschließen. Daher ist eine gewisse Vorsicht bei der Verwendung von makedepend angebracht.

[6] Merke: Mit make können auch andere Programme als nur der Compiler aufgerufen werden!

[7] Dies ist besonders zu berücksichtigen, falls sich Makefile gerade im Texteditor befindet und von dort aus make depend aufgerufen wird!

7 Definition und Anpassung eines Makefiles

Zur Definition und Anpassung eines Makefiles an ein eigenes Projekt müssen im wesentlichen die folgenden Vorbereitungen getroffen werden:

1. Definition (oder Änderung) eines Makros zur Bezeichnung der Quelldateien und Header:

   ```
   HEADERS = header1.h header2.h
   SRC     = file1.c file2.c file3.c
   ```

2. Entsprechend für die Objektdateien:

   ```
   OBJS = file1.o file2.o file3.o
   ```

3. Falls erforderlich müssen die Macros INCLUDE, LIBDIRS und LIBS mit den benötigten Verzeichnissen und Bibliotheken definiert werden, z.B.:

   ```
   INCLUDE = -I/home6/public/fruend/swt
   LIBDIRS = -L~/mylibs
   LIBS    = -lgks
   ```

   Damit wird die Bibliothek `libgks.a` aus dem Verzeichnis `$HOME/mylibs`[8] mit eingebunden. Die zugehörigen Header-Dateien werden zusätzlich zu den vom System vorgegebenen Systemverzeichnissen im Verzeichnis /home6/public/fruend/swt gesucht.

   Da auch die einzelnen Libraries von einander abhängig sein können, ist die Reihenfolge der einzelnen Bibliotheken **nicht** beliebig variierbar. Der Compiler wertet die einzelnen Dateien von links nach rechts aus.

4. Hinzufügen der Definition des gewünschten Ziels (meist der Name des ausführbaren Programms), z.B.:

   ```
   exefile:      $(OBJS)
                 cc -o $@ $(OBJS) $(LIBDIRS) $(LIBS)
   ```

   Dazu müssen im Eintrag für ein Ziel nur drei Änderungen vorgenommen werden: Eintrag des neuen Ziels und Anpassung des $(OBJS) Makro in Abhängigkeiten und Regel. ($@ steht hier als Platzhalter für den Namen des Ziels.)

5. Nach Aufruf von make depend kann mit make exefile das neue Ziel erzeugt werden. make depend muß immer dann neu aufgerufen werden, falls sich Änderungen an der #include-Struktur der Quell- und Header-Dateien ergeben haben.

Falls die Objekt-Dateien nicht mittels der voreingestellten Regeln erzeugt werden können, müssen noch entsprechende Einträge für deren Erzeugung ergänzt werden.

[8] Die Tilde `~` oder die Environmentvariable `$HOME` sind Platzhalter für den kompletten Pfadnamen Ihres Homedirectory.

8 Weitere Ziele im Beispiel-Makefile

```
# Erzeugen einer Funktionsbibliothek
libfourier.a: rmath.o fourier.o
        ar r libfourier.a rmath.o fourier.o

# Erzeugen des Testprogramms mittels Library
tf_lib:   libfourier.a rfkt.o tf.o
        cc -g -o tf_lib tf.o rfkt.o -L. ${LIBDIRS} -lfourier ${LIBS}

# Mit 'make clean' das Verzeichnis aufraumen
clean:
        /usr/bin/rm -f *.o *%
```

An diesen Zielen soll der Aufruf weiterer Programme über das Makefile verdeutlicht werden.

- Mit make lib wird eine Funktionsbibliothek mit dem Namen `libfourier.a` erzeugt. Diese Bibliothek kann jetzt anstelle der beiden Objekt-Dateien `rmath.o` und `fourier.o` zum Linken des ausführbaren Programms benutzt werden. Dazu ändert man das Ziel zum Erzeugen von tf wie folgt ab:

```
tf:         libfourier.a rfkt.o tf.o
        cc -g -o tf tf.o rfkt.o -L. ${LIBDIRS} -lfourier ${LIBS}
```

- Mit dem Ziel `tf_lib` ist die Erzeugung des Testprogramms unter Verwendung der neuen Fourier-Library realisiert.
- Sinnvoll ist auch ein Ziel, mit dem das Verzeichnis von temporären Dateien bereinigt werden kann. Es hat sich zu diesem Zweck ein Ziel names `clean` eingebürgert. Man verwendet hier den alsoluten Pfad zum Kommando rm, um nicht das Löschen jeder einzelnen Datei bestätigen zu müssen.

9 Weiterführende Literatur

A. Oram & S. Talbott *Managing Projects with make* **1991**, O'Reilly & Associates,Inc.

SUN Microsystems SUN Workshop 4.0 Answerbook

SUN Microsystems Manual-Page zu make