C/C++ Programmieren

Bei diesem Artikel handelt es sich um etwas gekürzte Kapitel aus dem ADIM Band 81: Linux

August Hörandl

In den folgenden Kapiteln werden der Compiler **gcc** bzw. **g++**, **make** und **gdb** direkt von der Kommandozeile aufgerufen. Es ist aber möglich, diese Programme, ähnlich einer integrierten Entwicklungsumgebung, direkt aus Emacs zu starten.

Viele Links zum Thema Programmieren (auch einige gut Tutorials) findet man z.B. unter
http://links.ee.htlw16.ac.at/linux/Programmieren/.

Der GNU C/C++ Compiler

Der GNU C/C++ Compiler hat ein modulares Konzept: er besteht aus drei Teilen:

- **Frontend**: Übersetzt in einen maschinen- und sprachunabhängigen Zwischenkode (RTL)
- **Optimizer**: ein eigenständiger Programmteil - damit können neue Optimierungen sehr leicht implementiert werden
- **Backend**: Übersetzt den Zwischenkode in Assembler für den entsprechenden Prozessor und erzeugt daraus die Opbjektdateien.
- Frontends gibt es für C, C++, Objective C, Pascal, Ada, Fortran etc.

Crosscompiling: der Compiler kann Code für fast jedes andere Betriebssystem bzw. fast jede andere unterstützte CPU erzeugen.

Derzeit gibt es insgesamt etwa 50 Prozessoren und 125 Host-Target (Crosscompiler) Kombinationen.

Editieren

siehe PCNEWS-60: Seite 52

Aufruf des Compilers

Nach dem Erstellen des Source Textes (**main.C**) wird der Compiler aufgerufen:

```
$ g++ -Wall -g main.C -O2 -o main
```

-Wall	viele Warnungen einschalten
-g	erzeuge Debuginfo
-o file	Name des Ergebnisses (ansonsten heißt das ausführbare Programm **a.out**)
-ON	Optimierung: der Wert von N bestimmt den Grad der Optimierung (0-keine bis 6-maximal, derzeit sind die Werte 3-6 gleichwertig)

gcc bzw. **g++** sind nur Steuerdateien. Sie rufen den eigentlichen Compiler, Assembler und Linker auf. Die Entscheidung wird dabei aufgrund der Endung getroffen:

.c .C	C Kode
.cpp	C++ Kode
.s	Assembler
.o	Objektdateien - nur linken

Makefile

Da dieser einzelne Aufruf bei größeren Projekten sehr umständlich und fehleranfällig ist und nicht immer alle Dateien neu übersetzt werden müssen, gibt es ein Programm zur Automatisierung dieser Arbeitsschritte: **make**

In der Steuerdatei (Standard: **Makefile**) stehen Regeln, z.B. wie eine Objektdatei aus C++ Programmtext erzeugt wird (die Anwendung von make ist dabei aber nicht auf Programmiertätigkeiten beschränkt).

Die Regeln haben die allgemeine Form

```
Ziel: Abhängigkeiten
  Kommandos
```

Ziel	was soll erzeugt werden
Abhängigkeiten	welche Dateien sind für die Erzeugung notwendig
Kommandos	welche Kommandos sind auszuführen

Zu einem Ziel kann es mehrere Regeln mit Abhängigkeiten geben.

Achtung

Vor den Kommandos muss ein Tabulator stehen - keine Leerzeichen!

Beispiel

Projekt mit 2 Dateien (**main.C stack.C**) und einer gemeinsamen Headerdatei (**stack.h**) ergibt sich damit folgendes Makefile:

```
main.o: main.C stack.h
  g++ -Wall -g -c main.C

stack.o: stack.C stack.h
  g++ -Wall -g -c main.C

main: main.o stack.o
  g++ stack.o main.o -o main
```

Die Datei **main.o** ist von der Datei **main.C** und von **stack.h** abhängig, d.h. wenn **main.o** nicht existiert bzw. älter als **main.C** oder **stack.h** ist, müssen die entsprechenden Kommandos ausgeführt werden.

Das Hauptprogramm **main** ist von den beiden Objektdateien abhängig.

Das Programm **make** erkennt auch die indirekten Abhängigkeit zwischen **main** und **stack.h**.

Beim Aufruf von **make** muss das zu erzeugende Ziel angegeben werden, ansonsten wird das erste Ziel erzeugt:

```
$ make main
```

Fehlersuche

Debugger - gdb

Der GNU Debugger heißt **gdb** und wird über eine einfache Kommandozeile bedient. Natürlich gibt es Completion und eine Commandhistory.

Auch unter X gibt es einige Oberflächen zum **gdb: xgdb, kdgb**. Der Debugger kann auch direkt von Editor Emacs aufgerufen werden; die Anzeige der aktuellen Position während der Ausführung erfolgt dann durch eine Markierung in der entsprechenden Datei.

Damit der Compiler Informationen über die Namen der Variablen, Funktionen usw. in das ausführbare Programm einfügt, muss **gcc** bzw. **g++** mit der Option **-g** aufgerufen werden.

```
gustl@goedel:~/stack > gdb main
GDB is free software and you are welcome to
distribute copies of it under certain
conditions;
type `"show copying`" to see the conditions.
There is absolutely no warranty for GDB;
type `"show warranty`" for details.
GDB 4.16 (i486-unknown-Linux
  -target i486-Linux),
Copyright 1996 Free Software Foundation, Inc.
(gdb)
```

Setzen eines Breakpoints am Beginn des Hauptprogramms und Start des Programms

```
(gdb) break main
Breakpoint 1 at 0x80486b6:
  file main.C, line 9
(gdb) run
Starting program: /home/gustl/stack/main
Breakpoint 1, main () at main.C:9
9 stack s;
(gdb)
```

Bei der Angabe der Befehle sind auch Abkürzungen möglich; n reicht für next (nächste Zeile):

```
(gdb) next
12 cout << `enter sentence ( . to end): `;
(gdb) n
13 cin.unsetf(ios::skipws);
(gdb) n
14 for(cin>>ch; cin && ch!='.'; cin>>ch)
(gdb) n
enter sentence ( . to end): abcdefg.
15 s.push(ch);
(gdb)
```

der Wert einer Variablen (auch Felder und Strukturen bzw. Klassen) kann mit dem Befehl print ausgegeben werden:

```
(gdb) print ch
$1 = 97 'a'
(gdb) p s
$2 = {
   buff = `äN ... 000000`, index = 0}
(gdb)
```

Der Debugger **gdb** bietet sehr viele Möglichkeiten:

- Einzelschritt (eine Zeile im Quelltext bzw. eine Assembleranweisung)
- Variablen ausgeben und ändern
- Direkter Zugriff auf die Register des Prozessors
- Breakpoints: Programm läuft bis zu einer bestimmten Zeile
- bedingte Breakpoints: Programm wird in einer bestimmten Zeile unterbrochen, wenn eine zusätzliche Bedingung erfüllt ist
- Watchpoints: Programm läuft bis ein gewisses Ereignis eintritt z.B. eine Variable erreicht einen bestimmten Wert
- Analyse von laufenden Programmen: der Debugger kann mit einem bereits laufenden Programm verbunden werden
- Analyse von **core** Dateien: diese können beim "Absturz" automatisch erzeugt werden, damit ist die nachträgliche Analyse eines Fehlers möglich (siehe Befehl **ulimit -c** bei **man bash**)
- Änderungen des Programmcodes (in Assembler) können direkt in die ausführbare Datei geschrieben werden (patch).

Entwicklungsumgebungen

Gerade am Gebiet der Entwicklungsumgebungen hat sich in letzter Zeit sehr viel getan. Unter *http://links.ee.htlw16.ac.at/linux/Programmieren/IDE_+_Editoren/* findet man einige Links zu solchen IDEs. Im Folgenden sollen einige davon näher vorgestellt werden.

Xwpe

Es handelt sich dabei um eine Entwicklungsumgebung ähnlich Borland C++ oder Turbo Pascal. Die Menüanordnung und Tastenbelegung ist eng an diese Vorbilder angelehnt.

Es können verschiedene Compiler und Linker eingesetzt werden. Im Fehlerfall kann die entsprechende Stelle im Sourcecode schnell angesprungen werden. Dabei werden auch Projekte mit mehreren Dateien unterstützt. Zur Fehlersuche können verschiedene Debugger direkt aus der Entwicklungsumgebung bedient werden.

Der große Vorteil von Xwpe liegt sicherlich in der Einsatzmöglichkeit auf einem Textschirm. Damit ist es z.B. für den einführenden Programmierunterricht an Schulen sehr gut geeignet.

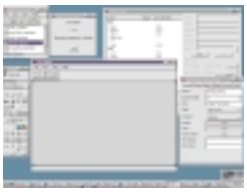
KDevelop & Glade

Auch die beiden neuen grafischen Oberflächen KDE und Gnome enthalten Entwicklungsumgebungen für den schnellen Entwurf (RAD) von grafischen Benutzerschnittstellen.

KDevelop

KDevelop (*http://www.kdevelop.org/*) ist ideal für die Programmentwicklung in C++ mit der Qt-Bibliothek. Neben der eigentlichen Programmentwicklung bietet es dem Programmierer viel Unterstützung durch umfangreiche Dokumentation. Ein Class-Browser zur besseren Übersicht bei großen C++ Projekten ist ebenfalls enthalten.

Glade



Glade (*http://glade.pn.org/*) ermöglicht die Programmentwicklung in C mit Unterstützung der GTK Bibliothek. Glade kann aber auch C++, Ada95, Python oder Perl Quellkode erzeugen.

SNiFF+ Penguin IDE

Unter *http://www.takefive.com/penguin/* gibt es eine freie Version der kommerziellen Entwicklungsumgebung SniFF. Dieses österreichische Produkt läßt bei der Programmentwicklung fast keinen Wunsch mehr offen: Editieren, Compilieren und Debuggen unter einer grafischen Oberfläche. Spezielle Tools um auch bei großen Projekte die Übersicht zu behalten z.B. Class-Browser.