Compilation – Software selber compilieren

Manuel Oetiker

16. Januar 2004

Vorbemerkung

In diesem Kurs wird das Kompilieren von Software behandelt, die als Quellcode (Source) verfügbar ist. Dieser Kurs behandelt die häufigsten Makesysteme und gibt Hinweise wie bei Fehlermeldungen vorgegangen werden kann. Es gibt viele verschiedene UNIX Systeme, ich werde speziell auf unser (Solaris 8) Environment eingehen wobei das Verfahren auf allen UNIX/LINUX Systemen ähnlich ist.

1 Einleitung

Die meisten Programme haben ein `configure` Shellskript das ein `Makefile` erzeugt. Mit Hilfe des Makefiles kann dann anschliessend mit dem Befehl `gmake` das Programm übersetzt werden. Die meisten der Programme nehmen an, dass die Befehle die sie zur konfiguration verwenden GNU[1] Programme sind. Wir haben unter Solaris diesen Programmen einen **g** prefix gegeben z.B. **g**make, **g**cat, **g**sed, **g**grep, **g**cc.... Unter linux sind diese programme fast immer GNU programme und meistens one **g** geschrieben.

1.) Schaue auf dem system nach, welche Compiler installiert sind. Du kannst unter `http://computing.ee.ethz.ch/sepp/report-date.html` Informationen finden. Welche Version hat der `gcc`, welche Version haben die anderen Compiler?

2.) Hole dir vom Internet das Programm **wget** Vers 1.9.1 pake es mit `man gtar` aus und kompiliere es so, dass in deinem Home `$HOME/prog` alles installiert wird. Lese zuerst das INSTALL im `wget-1.9.1` Verzeichnis. Du kannst es mit `less INSTALL` anschauen. Benutze `gmake` nicht wie beschrieben `make`. Unter Solaris ist gmake die GNU Version und make die Solaris make Version. Die Openssl-Library kannst du einbinden mit der Option `--with-ssl=/usr/pack/openssl-0.9.7c-ke/sun4u-sun-solaris2.8`

3.) Ändere deine `PATH` Variable so, dass du dein neues wget in `prog/bin` ohne absoluten Pfad benutzen kannst. Du kannst mit `which wget` überprüfen welches `wget` benutzt wird.

[1] `http://www.gnu.org`

4.) Wie könntest du es Anstellen, dass beim compilieren nicht der `gcc` sondern der `gcc-3.2` verwendet wird? (Shell variablen: CC C-Compiler, CXX C++-Compiler, F77 Fortran 77 Compiler, F95 Fortran 95 Compiler)

5.) Wie könntest du eine andere `gmake` version verwenden?

6.) Welche `configure` Option könntest du verwenden wenn du die Plattform abhängigen Programme in einem separates Verzeichnis installieren möchtest?

2 Compilieren und Linken

Leider läuft nicht immer alles so fehlerfrei wie beim wget Programm. Nicht alle Programmierer konzipieren ihr Make-System so, dass es auf mehreren Plattformen funktioniert.

Wir behandeln einige Fehler, die Auftreten können beim Compilieren der neusten Version von `lavaps`.

`-I/path/to/include` diese Option sagt dem Compiler, wo die include Dateien liegen. Das sind alle Header Files, z.B. `tk.h`

`-L/path/to/libraries` diese Option sagt dem Compiler, wo die library Dateien liegen. Das sind alle Library Files mit der Endung für dynamisch `.so.N.N` oder statisch `.a`.

`-R/path/to/libraries`[2] diese Option speichert den Path, wo die Libraries sich befinden, in das Programm, so dass das Programm beim Starten weiss, wo sich die Library im System befindet. Dies ist nur bei dynamisch gelinkten Programmen nötig. Wenn der Path zu den Libraries nicht im Binary gespeichert ist, gibt es noch die Möglichkeit, die Umgebungsvariable `LD_LIBRARY_PATH` zu setzten. Diese hat den Nachteil, dass dieser Path Priorität hat und es zu Versions–Konflikten kommen kann, wenn Programme verschiedene Library Versionen benötigen.

`-llib` diese Option sagt dem Compiler, dass er diese library linken soll. Wenn z.B. `libtk8.3.so` gelinkt werden soll ist die Option `-ltk8.3`. Der Prefix `lib` und die Endung `so.N.N` wird nicht geschrieben.

7.) Kopiere die lavaps `/home/moetiker/lavaps-2.3.tar.gz` in dein home und untare es.

8.) Starte das Skript `configure` mit der Option `--help`. Probiere die richtigen Optionen zu finden, damit das Programm in unserem Environment compiliert. Die librarys befinden sich unter `/usr/pack/tcltk-8.3.4.solaris-mo`. Benutze auch die Option `--prefix` wie bei `wget`.

9.) lavaps könnte auch noch mit einem andern GUI[3] Toolkit kompiliert werden. Das würde bei uns mit folgender Zeile gehen.
Ein \ die Zeile geht auf der nächsten Zeile weiter.

[2] für linux (`-Wl,-path,/path/to/libraries`)
[3] graphical user interface

```
wget http://www.isi.edu/~johnh/SOFTWARE/LAVAPS\
     /lavaps-2.3.tar.gz

gtar xfvz lavaps-2.3.tar.gz

cd lavaps-2.3

env PATH=/usr/pack/gnome-2.4.1-mo/sun4u-sun-solaris2.8\
     /bin:${PATH} \
PKG_CONFIG_PATH=/usr/pack/gnome-2.4.1-mo\
     /sun4u-sun-solaris2.8/lib/pkgconfig \
     CC=gcc-3.2.3 CXX=g++-3.2.3 \
./configure --with-gtk --prefix=$HOME/prog/lavaps

gmake install
```

10.) Überprüfe mit dem Befehl `ldd ~/prog/bin/lavaps` welche Libraries mit dem Programm dynamisch gelinkt sind.

Wenn Du etwas an den Optionen von `configure` geändert hast kannst du es einfach noch mal starten. Dadurch wird ein neues Makefile erzeugt. Im Makefile sind normalerweise die Optionen `install, clean, distclean` implementiert. Es gibt aber auch da kein Muss. Du kannst die Optionen nur rausfinden wenn du im Makefile nachschaust. Mit `gmake install` wird nur der Installations–Prozess gestartet. Mit `gmake clean` werden alles ausser die Konfiguration gelöscht. Dies kannst du nach einer Änderung der Konfiguration starten. Mit `gmake distclean` wird alles gelöscht ausser den Dateien, die in der Distribution (tarfile) sind.

11.) Starte die gmake Option und schaue was passiert. Mit `gmake -n [option]` wird nur angezeigt was gmake machen wird.

3 Environment Variablen

Durch das configure Skript werden Variablen im Makefile gesetzt, die dann anschliessend von gmake beim Übersetzen des Programms benötigt werden. Diese Variablen lassen sich beim Aufruf von gmake auch überschreiben
z.B. `gmake CC=gcc-3.2.3 CXX=g++-3.2.3`. Bei diesem Aufruf wird die Variable CC/CXX durch den Wert gcc-3.2.3 ersetzt, dies hat die Wirkung, dass im Ersatz zum `gcc` der Compiler `gcc-3.2.3` benutzt wird.

CFLAGS	Compiler flags (`-g -O3 -mcpu=ultrasparc`)
LDFLAGS	Solaris (`-L/Path/to/file -R/Path/to/file`) Linux (`-L/Path/to/file -Wl-rpath,/Path/to/file`)
CPPFLAGS	flags für CPP (`-I/usr/include`)
CC	C Compiler (`gcc-3.2.3`)
CXX	C++ Compiler (`g++-3.2.3`)
LD_LIBRARY_PATH	Path zu den Libraries (-L)
LD_RUN_PATH	Path zu den Libraries (-R)
MAKE	Programmname für make (`make`)
GNUMAKE	Programmname für gnumake (`gmake`)
PKG_CONFIG_PATH	Path zu den libraries config files (`gtk+-2.0.pc`)

Im Makefile gibt es viele weitere Variablen die verwendet werden. Es hängt auch zusätzlich vom Makesystem ab, inwiefern das Setzten einer solchen Variable etwas verändert.

12.) Schaue dir das Makefile von lavaps an und suche ein paar solcher Variablen.

4 Software Gruppen

Es gibt verschiedene Gruppen von OpenSource Produkten:

KDE KDE Applikationen nötigen meist die Variable `KDEDIR`. KDE hat sehr viele Libraries die gebraucht werden. Bei uns ist KDE unter `/usr/pack/kde-3.2.1-to` installiert. Du kannst im File `$KDEDIR/SEPP/INSTALL` nachschauen wie dieses Paket installiert wurde.

GNOME GNOME Applikationen benutzen wie KDE eine grosse Menge von Libraries. GNOME Desktop läuft noch nicht gut aber die Libraries sind Compiliert unt unter `/usr/pack/gnome-2.4.1-mo` installiert.

GNU Applikationen von `http://www.gnu.org` oder ETH mirror `ftp://sunsite.cnlab-switch.ch/mirror/gnu` haben meist ein configure make system das auch auf Solaris läuft. Gnu Software ist in der Regel sehr portabel programmiert.

QT Applikationen die den QT Toolkit von Trolltech verwenden. Hier wird normalerweise die Variable `QTDIR` erwartet. Die aktuelle QT Version 3.2.0 ist mit `gcc-3.2.3` compiliert.

GTK Applikation die den GTK2 (Gimp Toolkit) verwenden benutzen meistens das Script `pkg-config`. Bei älteren Programmen wird `gtk-config` verwendet. Mit diesen Skripts findet das Makesystem automatisch wo sich die gtk Bibliotheken befinden.

Grafik Viele Grafische Programme benutzen eine Grafik Bibliothek (Z.B. jpeg, gif, png) diese Bibliotheken sind in einem SEPP Paket zusammen gefasst. `gfxlibs-1.0-ds`.

Es gibt auch Programme die mit einem `Imakefile` konfiguriert werden. Hier wird aus dem Imakefile mit `xmkmf -a` ein Makefile generiert. Dann gibt es noch Programme die ihr eigenes Konfigurationsystem mitbringen. z.B. Ein File, in dem die Werte wie Prefix gesetzt werden können.

5 Errors beim Compilieren

Es ist wichtig, alle Beschreibungen und Anweisungen die mit der Source mitgeliefert werden, zu lesen. Speziell ist darauf zu achten, welche Versionen der Librarys oder Toolkits nötig sind, um das Programm zu übersetzen.

Bei Fehlern während des Ablaufs von configure kann im File `config.log` nachgeschaut werden was configure gemacht hat. Als weiteren Schritt kann das configure Shellskript mit `sh -x ./configure` gestartet werden. Mit dieser option wird jede Zeile angezeigt bevor sie ausgeführt wird.

13.) Mache im lavaps Verzeichnis ein `gmake distclean`, lasse configure laufen und schaue dir config.log an.

14.) starte das configure Script mit der Option `sh -x`. Was fällt auf?

Library not found: Dies weist darauf hin, dass die Library libtk8.3.so nicht gefunden wurde. Es sollte irgendwo noch ein `-L/path/xx -R/path/xx` eingefügt werden.

```
ld: fatal: library -ltk8.3: not found
ld: fatal: library -ltcl8.3: not found
ld: fatal: File processing errors. No output written to lavaps
collect2: ld returned 1 exit status
gmake[2]: *** [lavaps] Error 1
```

ld.so.1: Wenn das in der Fehlermeldung steht ist sehr wahrscheinlich ein Programm aufgerufen worden, dass seine Library selber nicht finden kann. z.B. msg2qm findet libqt nicht. In solchen Fällen kann mit einem `LD_LIBRARY_PATH` nachgeholfen werden.

```
/usr/pack/qt-2.1.1-to/solaris/bin/msg2qm ./tr.po ../po/tr.qm
ld.so.1: /usr/pack/qt-2.1.1-to/solaris/bin/msg2qm: fatal: libqt.so.2:
      open failed: No such file or directory
Killed
gmake[2]: *** [all-local] Error 137
```

bad value switch: Mit dieser Meldung ist meist eine Compiler Option im Spiel, die von dieser Compiler Version nicht verstanden wird.

```
c++ -DHAVE_CONFIG_H -I. -I. -I.   -g0 -O2 -mcpu=gugus
 -I/usr/pack/tcltk-8.3.2.solaris-mo/include  -I/usr/openwin/include
```

```
 -DUSE_PROCESS_SCAN_SOLARIS -g  -DUSE_TCL_BLOB  -Wall
 -DLAVAPS_STL_NAMESPACE=none -I/usr/openwin/include
 -c process_scan_bsd.cc
cc1plus: bad value (gugus) for -mcpu= switch
```

Undefined symbols: Wenn so etwas auftritt, sind in den Objekt files Symbole enthalten, die nicht gefunden werden. Dies bedeutet, dass sie mittels einer dazugelinkten Libary zur Verfügung gestellt werden müssen. `-llibrary`

```
Undefined                       first referenced
 symbol                             in file
Tcl_Eval                            tcl_blob.o
Tcl_SetResult                       tcl_blob.o
Tcl_CreateCommand                   tcl_blob.o
Tcl_Init                            tcl_blob.o
Tk_Init                             tcl_blob.o
```

Fehlerhafter Code Es kann natürlich immer sein, dass im Source-Code Fehler stecken. Da hilft nur eines: den Code anschauen und flicken oder einen Bugreport an den Entwickler zu senden.

6 Learning by doing

15.) Suche dir ein Programm das du gerne compilieren möchtest und Compiliere es.

7 Links und Information

GNU Software: `http://www.gnu.org`

Opensource Software: `http://freshmeat.net/`

MAKE: `http://www.gnu.org/manual/make/html_mono/make.html`

GCC: `http://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/`

SEPP ISG.EE: `http://computing.ee.ethz.ch/sepp` speziell sind hier die Links auf das `INSTALL` file. In diesem File beschreiben wir jeweils, was wir gemacht haben um die Source zu übersetzen.