Crosscompiling - Entwickeln für andere Systeme

Wolfgang Dautermann

FH JOANNEUM

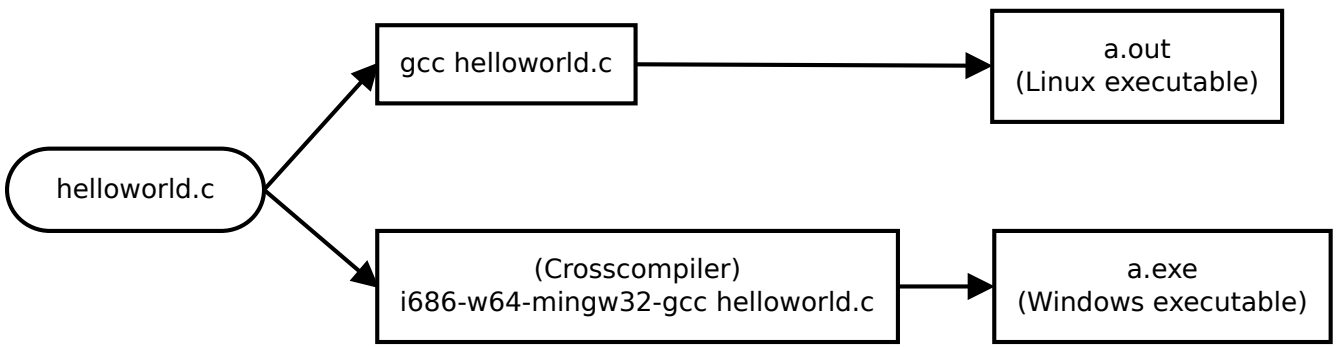
LIT Augsburg 2019

Über mich

- Sysadmin an der FH Joanneum
- erste Erfahrungen mit Crosscompiling vor mehreren Jahren
- 2014: CAS Maxima wird an der FH verwendet, Windows-Installer kommen nur selten daher
- $\Rightarrow$ Crosscompilieren des Maxima-Installers
  inklusive Abhängigkeiten
  Programmiert u.a. in Lisp

Cross-Compiler

Ein Cross-Compiler ist ein Compiler, der auf einer Plattform läuft, aber Compilate (Executables) für eine andere Plattform erzeugt.



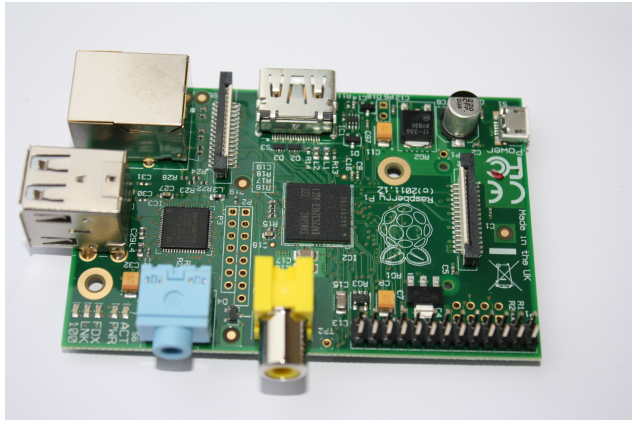
Cross-Compiler

Dabei können sich sowohl am Build-Rechner als auch auf der Zielplattform **alle** Komponenten unterscheiden:

- Betriebssystem
- CPU-Typ
- 32/64 Bit CPU (auch 16 oder 8 Bit)
- Big / Little Endian

Die Target-Plattform muss nicht mal real existieren!

Einsatzzwecke - Embedded Systeme



- Speicher/CPU-Ressourcen reichen für Compiler/Toolchain nicht aus
- Ein komfortable Entwicklungsumgebung läuft auf anderer Plattform
- Zeitersparnis: anderer Rechner hat mehr Ressourcen und kann schneller compilieren.

Einsatzzwecke

- Gewohnte Entwicklungsumgebung läuft nicht auf anderen Plattformen
- Build-Umgebung soll einheitlich sein
- Target-Plattform nicht verfügbar
- Softwarelizenz (Betriebssystem und/oder Entwicklungstools) nicht vorhanden
- (Noch) kein Compiler auf der Targetplattform verfügbar $\rightarrow$ Bootstrapping
- Kontinuierliche Integration – aktueller Softwarestand wird automatisch für diverse Plattformen compiliert, um Probleme früh zu erkennen.

Hello world unter Linux compilieren

hoffentlich bekannt...

Compilieren von helloworld.c

```
$ gcc -Wall -o helloworld helloworld.c
$ file helloword
helloworld: ELF 64-bit LSB executable, x86-64, [...]
```

Crosscompilieren mit GCC

Unterstützt etliche Architekturen und Programmiersprachen
(C, C++, Fortran, ObjC, ...)

Crosscompiler (unter Ubuntu)

```
apt-cache search ^gcc-8 | grep compiler
apt-cache search gfortran-8 | grep compiler
apt-cache search "g\+\+-8" | grep compiler
apt-cache search gobjc-8 | grep compiler
apt-cache search gccgo-8
apt-cache search ^gnat-8
```

Windows: Mingw

Minimale GNU Umgebung

Windows-Crosscompiler

```
apt-cache search mingw-w64

apt-get install g++-mingw-w64-i686
apt-get install g++-mingw-w64-x86-64
```

Hello world unter Linux **für Windows** compilieren

Cross-Compiler ist installiert unter `/usr/bin/i686-w64-mingw32-gcc`.
Einfach den speziellen Compiler verwenden:

Compilieren von helloworld.c

```
$ i686-w64-mingw32-gcc -Wall -o helloworld.exe helloworld.c
$ file helloword.exe
helloworld.exe: PE32 executable (console) Intel 80386, for MS Windows
```

erzeugt ein Windows-Executable.

Programme unter Linux **für Windows 64 Bit** compilieren

Compilieren von helloworld.c

```
$ x86_64-w64-mingw32-gcc -Wall -o helloworld.exe helloworld.c
$ file helloword.exe
helloworld.exe: PE32+ executable (console) x86-64, for MS Windows
```

erzeugt ein 64 Bit Windows-Executable.

Umfangreichere Programme unter Linux **für Windows** compilieren

Die notwendigen Bibliotheken (Libraries) müssen auch für die Zielplattform verfügbar sein[1]. Ansonsten kann ganz normal compiliert werden.

Compilieren von LibSDL + helloworld-sdl.cpp

```
./configure --prefix=/opt/libsdl --build=$(gcc -dumpmachine)
            --host=$(x86_64-w64-mingw32-gcc -dumpmachine)

$ x86_64-w64-mingw32-gcc -o helloworld-sdl.exe helloworld-sdl.c \
         $(/opt/libsdl/bin/sdl2-config --libs --cflags)
$ file helloworld-sdl.exe
helloworld-sdl.exe: PE32+ executable (GUI) x86-64, for MS Windows

$ cp /opt/libsdl/bin/SDL2.dll .
$ wine helloworld-sdl.exe
```

[1] entweder selbst übersetzen oder fertig downloaden

Crosscompilieren mit LLVM / Clang

Ist ein Crosscompiler für etliche Ziele.

Beispiel Clang

Zielangabe durch Compileroption

```
-target <triple>

Format:
<arch><sub>-<vendor>-<sys>-<abi>

        arch = x86_64, i386, arm, thumb, mips, etc.
        sub = for ex. on ARM: v5, v6m, v7a, v7m, etc.
        vendor = pc, apple, nvidia, ibm, etc.
        sys = none, linux, win32, darwin, cuda, etc.
        abi = eabi, gnu, android, macho, elf, etc.
```

Crosscompilieren mit LLVM / Clang

Live-Demo

- diverse CPUs
-  FreeBSD
- base.tgz wird benötigt
  `--sysroot=$(pwd)/freebsd-11.2`

Andere Compiler/Programmiersprachen

Beispiel Go

Etliche (z.B C, C++, ObjC, Fortran) durch GCC unterstützt).

Go: Zielangabe durch Umgebungsvariablen

```
go build helloworld.go
[...]
GOOS=windows go build helloworld.go
[...]
GOOS=linux GOARCH=arm go build helloworld.go
```

Andere Compiler/Programmiersprachen: Pascal

Nicht mehr ganz die Sprache, die in der Schule gelehrt wird...

Freepascal bietet ein nettes Tool (fpcupdeluxe) um Crosscompiler zu erstellen.

(Freepascal-Logo)

Andere Programmiersprachen

Beispiel Lisp

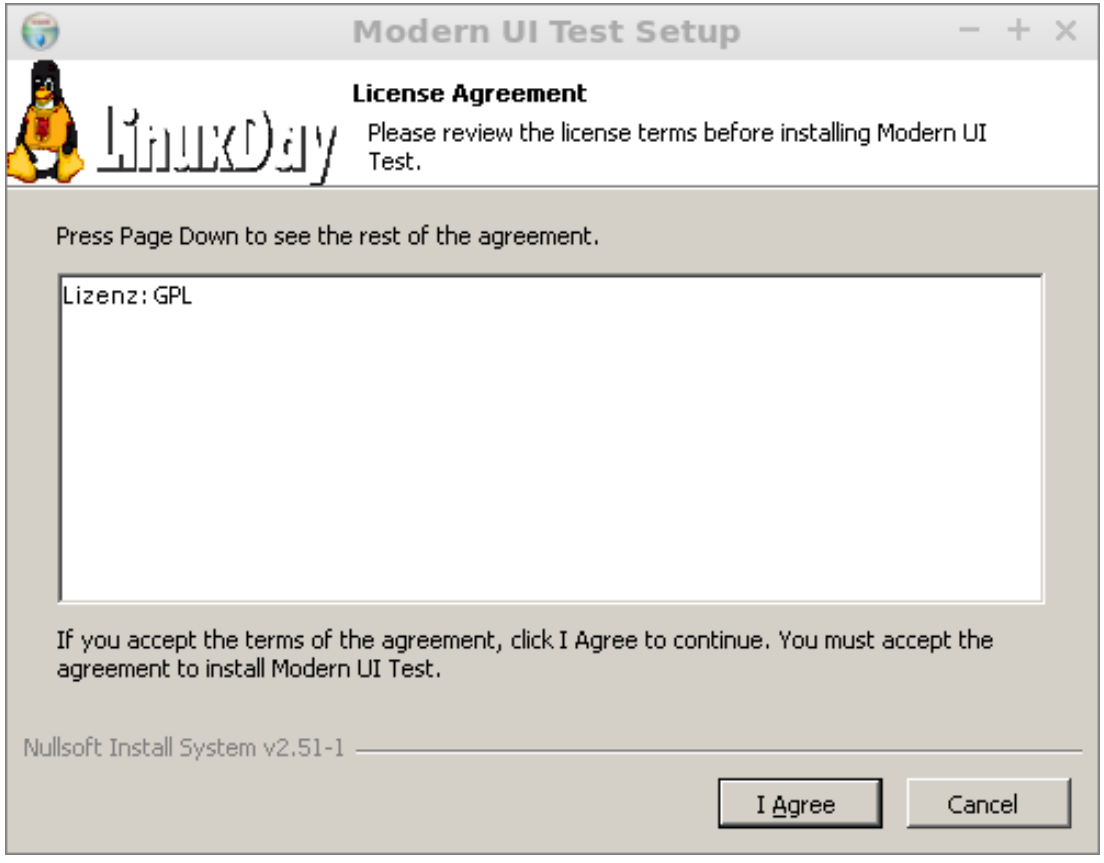
Crosscompiler nicht vorhanden. Lösung für den Maxima-Installer: Verwendung des Windows-Compilers mittels Wine.

Win-Installer extrahieren und Binary verwenden

```
7z x sbcl-1.4.2-x86-64-windows-binary.msi

wine64 ./sbcl.exe --core ./sbcl.core "$@"
```

Paketieren mit NSIS



Extrahieren existierender Windows-Installer

Extrahieren von Microsoft-Formaten

7z extrahiert MSI, EXE Installer

Auflisten des Inhalts:

```
$ 7z l installer.msi # oder exe
```

Extrahieren:

```
$ 7z x installer.msi # oder exe
```

cabextract extrahiert CAB-Archive

Extrahieren:

```
$ cabextract file.cab # --list file.cab zum Auflisten
```

Programme Crosscompilier-freundlich machen

Bitte bei eigenen Projekten berücksichtigen

- Keine hartcodierten Compiler (`gcc`) im Makefile
  Variablen, z.B. `$(CC)` verwenden!
- Keine Annahmen über Typgrössen. `sizeof(TYP)` verwenden
- Verzeichnistrenner: `/` (geht auch unter Windows)
- Plattformübergreifende Libraries verwenden (z.B. für Netzwerk, ...)
- Andere Compiler regelmässig verwenden.
- Crosscompiling selbst versuchen.

Vielen Dank

Fragen? (hoffentlich richtige. . .) Antworten!

Vielen Dank für Ihre Aufmerksamkeit

Wolfgang Dautermann

wolfgang.dautermann [AT] fh-joanneum.at