Multimedia Technieken

2015 – 2016

Cross-compiling Code for the Raspberry Pi 2

katholieke hogeschool vives
associatie KU Leuven

Based on http://hertaville.com/2012/09/28/development-environment-raspberry-pi-cross-compiler/

Cross-compiling Code for the Raspberry Pi 2

What is Cross-Compilation

katholieke hogeschool
associatie KU Leuven
vives

What is Cross-Compilation

- A native compiler such as the default gcc tool on the PC is a compiler that runs on an Intel machine, as well as creates binaries intended to be run on an Intel machine.
  - It creates binaries for the same type of machine that it runs on.
- Similarly the GCC tool in the RPi's Raspbian Linux OS is intended to run on an ARM machine as well as creates binaries for an ARM machine.

- A cross compiler such as the "arm-linux-gnueabihf-gcc" that we will use, is able to run on an Intel machine but creates binaries for an ARM machine.
  - It runs on one architecture and creates binaries for another.
  - This allows us to develop and compile our programs/libraries on our Desktop PC, but when it comes to deploying the binaries/libraries we deploy them and run them on the Raspberry Pi.

What is Cross-Compilation

- Why ?
  - The main reason to use cross-compilation over native compilation (develop and compile on the RPi itself) is to speed up compile/build time.
  - The RPi board may be fast compared to a microcontroller ... but its still has limited RAM resources and is pretty slow compared to an average desktop computer

- Also you have a myriad of development tools that you can use on your desktop PC that you simply can't use on the Raspberry Pi

- Cross compilation of simple applications that only require the use of the standard C/C++ libraries included with the cross compiling toolchain is relatively straightforward.
- Cross compilation of more complicated applications that require external libraries however (such as libjpeg, GTK+, Qt4/5 e.t.c), is typically more complicated to setup

katholieke hogeschool vives
associatie KU Leuven

Cross-compiling Code for the Raspberry Pi 2

Setting Up the Cross Compiling Toolchain

katholieke hogeschool vives
associatie KU Leuven

Basic Requirements

- To start developing on your host machine you will first need to install some basic packages

```
bioboost@NDWMINT ~ $ sudo apt-get install build-essential git
```

- This will install the native build tools on your PC along with the GIT tool which will be used to download / clone the cross compiling toolchain from GitHub.com.

- Clone the Raspbian's official cross compiling toolchain from Github

```
bioboost@NDWMINT ~ $ git clone
git://github.com/raspberrypi/tools.git rpi_tools
```

The Toolchain

- If you are running a 32-bit Linux operating system, you'll want to use the third toolchain 'gcc-linaro-arm-linux-gnueabihf-raspbian'.
- If however you are running a 64-bit Linux operating system, you'll want to use the fourth toolchain 'gcc-linaro-arm-linux-gnueabihf-raspbian-x64.
- If you don't know whether you are running 32-bit or a 64-bit version of Linux type in the command line

```
bioboost@NDWMINT ~ $ uname -a
```

  – If you see something like 'i386'/'386' in the output string, then you're running a 32-bit version of Linux. If however you see 'x86_64' / 'amd64' in the output string, then you're running a 64-bit version of Linux.

Cross-compiling Code for the Raspberry Pi 2

Makefiles

Compilation using Makefiles

- Makefile(s) are text files written in a certain prescribed syntax.
- Together with the Make utility, it helps build a software from its source files, a way to organize code, and its compilation and linking.

- Start by creating a really simple C++ program in your home dir

- Example:

```
#include <stdio.h>

int main(void)
{
        printf("Hello world\r\n");
        return 0;
}
```

- Save it as main.cpp

Compilation using Makefiles

- You can compile this program using the following command

```
$ gcc –o hello main.cpp
```

- Or we can make a really simple makefile

```
# The compiler
CC=gcc

# Compiler flags
CFLAGS=-Wall
  #  -Wall turns on most, but not all, compiler warnings


hello: clean
           $(CC) $(CFLAGS) -o hello main.cpp

clean:
           rm -f hello
```

- Save it as “Makefile”

Compilation using Makefiles

- Next you just execute the “make” command to generate your executable
- Or “make clean” to clean your output files

- Output:

```
$ ls
hello  main.cpp  Makefile
```

- Now you can run the program

```
./hello
Hello world
```

Cross-compiling using Makefiles

- Now we got a small program to compile on our host system but when we want to run this on the Pi we need to cross-compile it

- To make life easier you can use the following Makefile as a base for your work

```
# The compiler
BINPATH=/home/bioboost/rpi_tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-
CC=$(BINPATH)gcc

# Compiler flags
CFLAGS=-Wall
  #  -Wall turns on most, but not all, compiler warnings


hello: clean
            $(CC) $(CFLAGS) -o hello main.cpp

clean:
            rm -f hello
```

- You can download this from Toledo (make_hello)

Cross-compiling Code for the Raspberry Pi 2

Transferring Files to the Raspberry Pi

Transferring Files - Secure Copy

- To transfer files from one system to another (local or remote) we can use the secure copy tool which has the following syntax

```
bioboost@NDWMINT ~ $ scp [[user@]host1:]source
[[user@]host2:]destination
```

  – Scp allows us to copy files from one system to another over a secure SSH connection

- Example:
  – This will copy the hello executable to the root of the target system

```
bioboost@NDWMINT ~ $ scp hello root@172.180.35.10:/
```