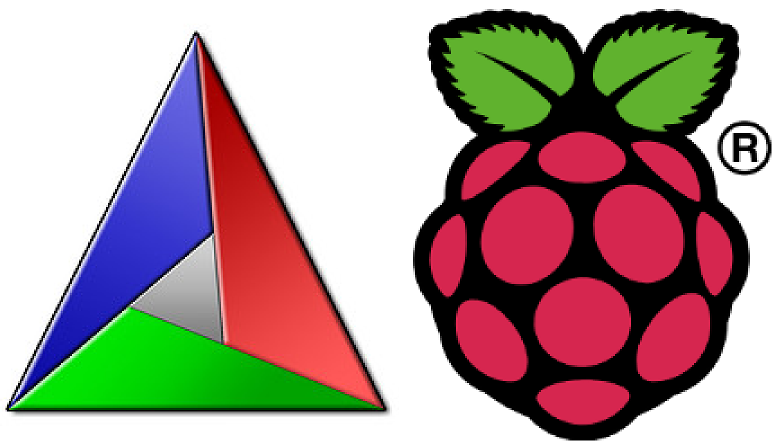
Cross-compiling & debugging for RaspberryPi using CMake and gdbserver/gdb

Sábado, 16 de marzo de 2013 piponazo Dejar un comentario Ir a comentarios



The Raspberry Pi (RPi in advance) board has become in the last year one of the favourite toys for geeks & programmers. Although it has a reasonable powerful processor, if we have in mind to develop a large project in this little board, the development process should be performed in a (more suitable) workstation and then perform cross compilations to generate binaries for the RPi architecture (ARM). There already exists some available tutorials explaining how to do the cross-compilation and debugging using the Eclipse IDE. However, as most of the readers of this blog know, I don't like to be tied to the usage of a specific IDE, and I have been using CMake for configuring my software projects for quite a while (and letting anyone using their favourite programming environment to develop). Therefore, in this post I will explain how to do the cross-compilation process with CMake and how to debug the applications in the remote target from our host workstation. Here we go!

Cross-compiling with CMake

I will assume that you know what CMake is (if not, you can take a look to this older post). The application that we want to compile is a simple "Hello World" example with a loop that counts from 0 to 99 sleeping the application one second in each iteration of the loop:

```
#include <stdio.h>
#include <unistd.h>

int main()
{
  unsigned int i=0;
  printf("Hello world!!\n");
  for (i=0; i<100; i++) {
      printf("Counter: %d\n", i);
      sleep(1);
  }
  return 0;
}
```

The source code of the project can be download from this link. In the root folder of the project is provided a *toolchain-rpi.cmake* file which contains the necessary setup for indicating to CMake how to compile with the toolchain of Raspberry Pi. This toolchain should be previously downloaded from here into your local machine, using **git**. Then, in order to ease the development process, we will add the following lines to our $HOME/.bashrc:

```
export RPI_ROOT=~/programming/rpi-toolchain/arm-bcm2708/arm-bcm2708hardf
export PATH=$RPI_ROOT/bin/:$PATH
```

Taking a look to the *toolchain-rpi.cmake* we can appreciate how RPI_ROOT environment variable is used to specify where all the binaries of the toolchain are:

```
01  INCLUDE(CMakeForceCompiler)
02
03  SET(CMAKE_SYSTEM_NAME Linux) # this one is important
04  SET(CMAKE_SYSTEM_VERSION 1)  # this one not so much
05
06  SET(CMAKE_C_COMPILER   $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
07  SET(CMAKE_CXX_COMPILER $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
08  SET(CMAKE_AR           $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
09  SET(CMAKE_LINKER       $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
10  SET(CMAKE_NM           $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
11  SET(CMAKE_OBJCOPY      $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
12  SET(CMAKE_OBJDUMP      $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
13  SET(CMAKE_STRIP        $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
14  SET(CMAKE_RANLIB       $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueab
15
16  # where is the target environment
17  SET(CMAKE_FIND_ROOT_PATH  $ENV{RPI_ROOT}/arm-bcm2708hardfp-linux-gnueabi
18
19  # search for programs in the build host directories
20  SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
21
22  # for libraries and headers in the target directories
23  SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
24  SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Now, we are able to perform the cross-compilation of our simple application. These are the steps for creating two build folders (one for RPi architecture and other for the host architecture):

```
user@host:~/$ cd path-to-rpi-project
user@host:~/rpi$ mkdir build_rpi
user@host:~/rpi$ mkdir build_host
user@host:~/rpi$ cd build_rpi
user@host:~/rpi/build_rpi$ cmake -D CMAKE_TOOLCHAIN_FILE=../toolchain-rpi.cmake
user@host:~/rpi/build_rpi$ make
user@host:~/rpi/build_rpi$ make install
user@host:~/rpi/build_rpi$ cd ../build_host
user@host:~/rpi/build_host$ cmake ..
user@host:~/rpi/build_host$ make
```

> ***Important***: I have added a nice feature in the CMake configuration of this project in order to mount automatically the $HOME folder of the RPi in our host machine using sshfs. Therefore, we will need to have **sshfs** package installed in our system. Setting in the CMake configuration the variable CMAKE_INSTALL_PREFIX to this path, we are able to copy the binaries directly to the RPi with the command **make install**. If you don't like or want this feature, comment or delete the lines 16-22. In this case you should copy the binaries to your raspberry pi using, for example, **scp** command.

Note that I have created two different build folders, one for the RPi cross-compilation process and other for the host machine one. If our code doesn't depend of specific hardware or features of the RPi we can first test the program in our host machine for speed up the development process.

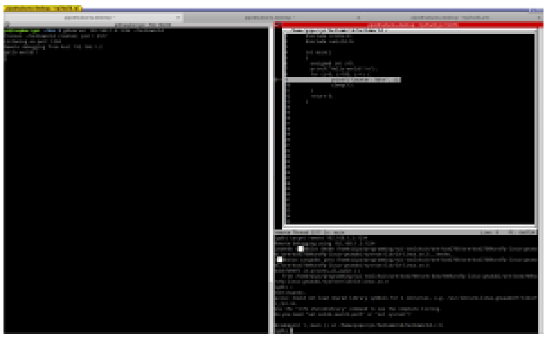
Cross-Debugging with gdbserver and gdb

Once installed the binary generated in the cross-compiling process to the RPi (either using the sshfs feature with **make install** command or using **scp**), let's see how we can debug remotely from our host workstation a program running in the RPi. For this purpose we need to run **gdbserver** in the RPi side (a read to the "**man gdbserver**" is very recommended). I will assume that we have connected both RPi and host workstation to the same local network, so we are going to listen connections in **gdbserver** using a TCP connection launching the application in the next way:

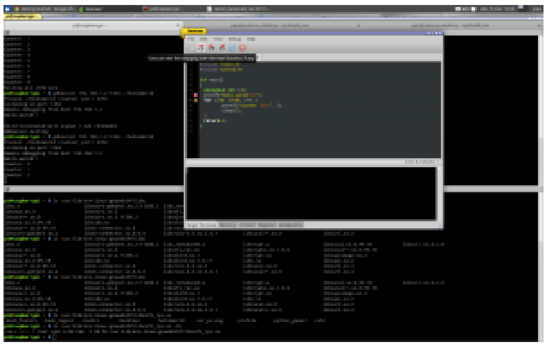
```
pi@raspberry:~/bin/$ gdbserver 192.168.1.2:1234 ./helloWorld
```

Then, in the host side, we are going to debug using **gdb**. We can't use the **gdb** host version of our host machine. We need to run the specific **gdb** version provided with the toolchain that is able to read the symbols for the architecture of the RPi (ARM). We can use both **gdb** or **gdbtui**, I recommend the second because of it shows the source code in a terminal window separated from the **gdb** command line. So, the steps in the host side are:

```
user@host:~/rpi/build_rpi$ arm-bcm2708hardfp-linux-gnueabi-gdbtui  hello
...
// Once inside gdb
(gdb) target remote 192.168.1.3:1234
(gdb) b 9
(gdb) c
...
```


Debugging with gdbserver & gdbtui remotely

If you prefer to use a Front-end for GDB, as **nemiver**, you only need to specify in the preferences of the application the gdb binary (from the RPi toolchain) and the remote target address and port while gdbserver is running in the RPi.


Debugging remotely using nemiver

Links:

http://hertaville.com/2012/09/28/development-environment-raspberry-pi-cross-compiler/ (Eclipse Tutorial)

https://github.com/raspberrypi/tools (toolchain repository)

http://www.unicom.com/blog/entry/651 (umount sshfs partitions)

★★★★★★★★★★
Rating: 10.0/**10** (4 votes cast)

Cross-compiling & debugging for RaspberryPi using CMake and gdbserver/gdb, 10.0 out of 10 based on 4 ratings

Share / Save

Categories: Hardware, Programación Tags: c++, camke, compilation, debug, gdb, gdbserver, git, nemiver, programming, raspberry pi
Comentarios (4) Referencias (0) Dejar un comentario Referencia

g+1 0

1. Jorge
Sábado, 14 de septiembre de 2013 a las 09:21 | #1
Responder | Citar
Rating: 0.0/**10** (0 votes cast)

Hi:
Great tutorial.
I have been able to compile for Rbpi but when I try to debug I am not able to see the code. Gdb says code unavailable.

Is there any flag to compile using debug info? What do I miss?

2. piponazo
Viernes, 20 de septiembre de 2013 a las 06:09 | #2
Responder | Citar
Rating: 0.0/**10** (0 votes cast)

Hi @Jorge .
Did you add the flags "-g -O0″ for compiling in debug mode ?

3. Jorge
Sábado, 21 de septiembre de 2013 a las 18:26 | #3
Responder | Citar
Rating: 0.0/**10** (0 votes cast)

No I didn't add, but where I should do it? I really sorry but it is the first time I use it.

4. frank71726
Jueves, 2 de abril de 2015 a las 10:45 | #4
Responder | Citar
Rating: 0.0/**10** (0 votes cast)

hi:

when I follow your steps and add a simple static library by myself, I get some errors as below.

```
frank@PC1657:~/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build$ make VERBOSE=1
/usr/bin/cmake -H/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace -
B/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build –check-build-system CMakeFiles/Makefile.cmake
0
/usr/bin/cmake -E cmake_progress_start /home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/CMakeFiles
/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/CMakeFiles/progress.marks
make -f CMakeFiles/Makefile2 all
make[1]: Entering directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
make -f src/CMakeFiles/RPI_LED_LIB.dir/build.make src/CMakeFiles/RPI_LED_LIB.dir/depend
make[2]: Entering directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
cd /home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build && /usr/bin/cmake -E cmake_depends "Unix
Makefiles" /home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace
/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/src
/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build
```

```
/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/src
/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/src/CMakeFiles/RPI_LED_LIB.dir/DependInfo.cmake
–color=
make[2]: Leaving directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
make -f src/CMakeFiles/RPI_LED_LIB.dir/build.make src/CMakeFiles/RPI_LED_LIB.dir/build
make[2]: Entering directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
Linking C static library libRPI_LED_LIB.a
cd /home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/src && /usr/bin/cmake -P
CMakeFiles/RPI_LED_LIB.dir/cmake_clean_target.cmake
cd /home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build/src && /usr/bin/cmake -E cmake_link_script
CMakeFiles/RPI_LED_LIB.dir/link.txt –verbose=1
"" cr libRPI_LED_LIB.a CMakeFiles/RPI_LED_LIB.dir/PJ_RPI.c.o
Error running link command: No such file or directory
make[2]: *** [src/libRPI_LED_LIB.a] Error 2
make[2]: Leaving directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
make[1]: *** [src/CMakeFiles/RPI_LED_LIB.dir/all] Error 2
make[1]: Leaving directory `/home/frank/hpbProject/RaspberryPi/rpi_prj/rpi_led_userspace/build'
make: *** [all] Error 2
```

If I comment SET(CMAKE_AR $ENV{RPI_ROOT}/bin/arm-bcm2708hardfp-linux-gnueabi-ar),
it can build successfully and running fine.

Could you give me some suggestions?

1. Sin trackbacks aún.

Bienvenido de nuevo **frank71726**. Cambiar »

Suscribirse a los comentarios

Publicar comentario (Ctrl+Enter)

Estás subscrito a esta entrada. Gestiona tus subscripciones.

OpenCV Computer Vision with Python Qt AUTOMOC with CMake
RSS

Twitter

Etiquetas

0

+1

My profile on Mendeley

Categories

- Breves
- Compiz
- Distribuciones
- Documentos
- Drivers
- Enlightenment
- General
- Gnome
- Gráficos
- Hardware
- Internet
- Investigación
- juegos
- KDE
- Latex
- Libros
- MSN
- Noticias
- Ocio
- OpenCV
- P2P
- Programación
- Redes
- Resolución problemas
- Scripting
- Seguridad
- sonido
- Tecnología
- Terminal
- Tutoriales
- Utilidades
- vim

Blogroll

- ArUco
- crystalxp.net
- Damiles
- El blog de Neonigma
- Entre tuxes y pepinos
- Hablando de linux
- Helektron
- Just 4 Cool!
- Phoenix Revolution
- Ubuntu Life

Últimos comentarios

- frank71726 en Cross-compiling & debugging for RaspberryPi using CMake and gdbserver/gdb
- Eduardo Lasso en Latex: Acrónimos
- LUIS MARIA en Beamer: Presentaciones profesionales en LaTeX
- Mrlinkin en LaTeX – Intercalado de imágenes en texto
- RSG en LaTeX – Intercalado de imágenes en texto

Valoración del blog

Average rating:

8.6

769 votes for **196** posts

Arriba WordPress