Create Cross-Toolchain for ARM (BBB) Assignment

Conduct the following steps to accomplish various tasks in completing this assignment.

Make sure the storage on the VM is large enough, if 25 GB of storage is not enough, increase 100 GB.

Note the $ indicate a system prompt and should not be entered as the start of a line with a command.

1. Update and Upgrade System

$ sudo apt update

$ sudo apt upgrade

2. Add Guest Addition to Virtualbox

Type command into terminal:

$ sudo apt install linux-headers-$(uname -r) build-essential dkms

$ sudo add-apt-repository multiverse

$ sudo apt install virtualbox-guest-dkms virtualbox-guest-x11

Click on VirtualBox Devices>Insert Guest Additions CD Image
CD should be added on bar on the left. Open it and right click and open it in terminal
In new terminal window, type: ./autorun.sh
Once complete, restart Virtual machine
Video followed: https://www.youtube.com/watch?v=zdkl16oAS1k

3. Toolchain: Install crosstool-ng

**$ sudo apt-get install automake bison chrpath flex g++ git gperf **

**gawk libexpat1-dev libncurses5-dev libsdl1.2-dev libtool libtool-bin **

python2.7-dev texinfo help2man

if it does not work then run individually

$ sudo apt-get install texinfo

$ sudo apt-get install gperf

$ sudo apt-get install libtool-bin

https://crosstool-ng.github.io/docs/

4. Make project directory for class

$ mkdir eel4734

$ cd eel4734

5. Obtain Crosstools from Github

$ git clone https://github.com/crosstool-ng/crosstool-ng.git

$ cd crosstool-ng

$ git checkout master

$ ./bootstrap

$ ./configure --enable-local

$ sudo make

$ sudo make install

To start crosstool-ng type:

$ ./ct-ng

(you should see information from the tool, indicating its version 1.25 and available commands)

6. Build a Tool Chain for the BBB

$ ./ct-ng list-samples

BBB has TI AM335x SoC

$ ./ct-ng show-arm-cortex_a8-linux-gnueabi

To Configure

$ ./ct-ng arm-cortex_a8-linux-gnueabi

$ ./ct-ng menuconfig

In Paths and misc options, disable Render the toolchain read-only (CT_INSTALL_DIR_RO)

In Target options, then Floating point, then select hardware (FPU) (CT_ARCH_FLOAT_HW)

"Target options --> Use specific FPU" and type "vfpv3" which is the floating point unit specification for the Arm Cortex A8.

$ ./ct-ng build

(now wait about an 1 hours)

(if it fails because it cannot retrieve a few tarball that it requires (site is down) you can do it manually.

Go to your home directory and create the src directory and add the following file, then return to the crosstool-ng directory and run the ./ct-ng build)

http://nweb.eng.fiu.edu/aperezpo/EEL4734/isl-0.24.tar.xz

This will create a directory called ~/x-tools/arm-cortex_a8-linux-gnueabihf

Toolchain location

~/x-tools/arm-cortex_a8-linux-gnueabihf/bin

$ PATH=~/x-tools/arm-cortex_a8-linux-gnueabihf/bin:$PATH

7. Test toolchain

Using gedit create a test program called helloworld.c that prints the phrase "Hello World"

$ arm-cortex_a8-linux-gnueabihf-gcc helloworld.c -o helloworld

$ file helloworld

(should see that it is a 32 bit ARM executable)

To get version number

$ arm-cortex_a8-linux-gnueabihf-gcc --version

To get the configuration details

$ arm-cortex_a8-linux-gnueabihf-gcc -v

Print out the range of arch-specific options available

$ arm-cortex_a8-linux-gnueabihf-gcc --target-help

The toolchain sysroot is a directory which contains subdirectories for libraries,

header files, and other configuration files

$ arm-cortex_a8-linux-gnueabihf-gcc -print-sysroot

lib: Contains the shared objects for the C library and the dynamic linker/loader, ld-linux

usr/lib, the static library archive files for the C library, and any other libraries that may be installed subsequently

usr/include: Contains the headers for all the libraries

usr/bin: Contains the utility programs that run on the target, such as the ldd command

usr/share: Used for localization and internationalization

sbin: Provides the ldconfig utility, used to optimize library loading paths

Command	**Description**
addr2line	**Converts program addresses into filenames and numbers by reading debug symbol tables in an executable file. It is very useful when decoding addresses printed out in a system crash report.**
ar	**The archive utility is used to create static libraries.**
as	**This is the GNU assembler.**
c++filt	**This is used to demangle C++ and Java symbols.**
cpp	**This is the C preprocessor and is used to expand #define, #include, and other similar directives. You seldom need to use this by itself.**
elfedit	**This is used to update the ELF header of the ELF files.**
g++	**This is the GNU C++ frontend, which assumes that source files contain C++ code.**
gcc	**This is the GNU C frontend, which assumes that source files contain C code.**
gcov	**This is a code coverage tool.**

gdb	**This is the GNU debugger.**
gprof	**This is a program profiling tool.**
ld	**This is the GNU linker.**
nm	**This lists symbols from object files.**
objcopy	**This is used to copy and translate object files.**
objdump	**This is used to display information from object files.**
ranlib	**This creates or modifies an index in a static library, making the linking stage faster.**
readelf	**This displays information about files in ELF object format.**
size	**This lists section sizes and the total size.**
strings	**This displays strings of printable characters in files.**
strip	**This is used to strip an object file of debug symbol tables, thus making it smaller. Typically, you would strip all the executable code that is put onto the target.**

Link math lib

myprog.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.14159265

int main () {
  double x, ret, val;

  x = 45.0;
  val = PI / 180;
  ret = sin(x*val);
  printf("The sine of %lf is %lf degrees", x, ret);

  return(0);
}
```

$ arm-cortex_a8-linux-gnueabihf-gcc myprog.c -o myprog -lm

(-lm means link libm , lib is not indicated)

See which lib linked with executable

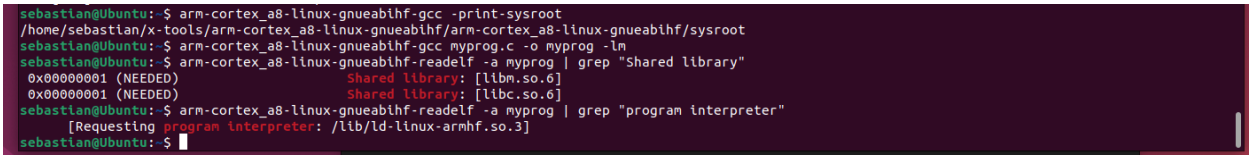
$ arm-cortex_a8-linux-gnueabihf-readelf -a myprog | grep "Shared library"

Shared libraries need a runtime linker, which you can expose using:

$ arm-cortex_a8-linux-gnueabihf-readelf -a myprog | grep "program interpreter"

(GENERATE Screenshot of the previous three statements and describe what the output generated by these commands means)

As below:



Static link

helloworld.c

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
   printf ("Hello, world!\n");
   return 0;
}
```

$ arm-cortex_a8-linux-gnueabihf-gcc helloworld.c -o helloworld

$ arm-cortex_a8-linux-gnueabihf-gcc -static helloworld.c -o helloworld-static

$ ls –l

(Describe why the size difference between helloworld and helloworld-static)

As below:

```
sebastian@Ubuntu:~$ ls -l
total 2804
drwxr-xr-x 2 sebastian sebastian    4096 May 27 22:06 Desktop
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Documents
drwxr-xr-x 2 sebastian sebastian    4096 May 27 20:39 Downloads
drwxrwxr-x 3 sebastian sebastian    4096 May 28 20:28 eel4734
-rwxrwxr-x 1 sebastian sebastian   12444 May 28 22:27 helloworld
-rw-rw-r-- 1 sebastian sebastian     117 May 28 22:27 helloworld.c
-rwxrwxr-x 1 sebastian sebastian 2778292 May 28 22:27 helloworld-static
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Music
-rwxrwxr-x 1 sebastian sebastian   12564 May 28 22:20 myprog
-rw-rw-r-- 1 sebastian sebastian     228 May 28 22:20 myprog.c
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Pictures
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Public
drwx------ 4 sebastian sebastian    4096 May 27 14:36 snap
drwxrwxr-x 3 sebastian sebastian    4096 May 28 20:36 src
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Templates
drwxr-xr-x 2 sebastian sebastian    4096 May 25 19:48 Videos
drwxrwxr-x 3 sebastian sebastian    4096 May 28 20:34 x-tools
sebastian@Ubuntu:~$
```

Static linking pulls code from a library archive, usually named lib[name].a. In the preceding case, it is libc.a, which is in [sysroot]/usr/lib:

$ export SYSROOT=$(arm-cortex_a8-linux-gnueabihf-gcc -print-sysroot)

$ cd $SYSROOT

$ ls -l usr/lib/libc.a

Now create your own static library called libtest.a

test1.c

```
#include <stdio.h>
void foo1_1(void)
{
  printf("Test1 – foo1\n");
}
void foo1_2(void)
{
  printf("Test1 – foo2\n");
}
```

test2.c

```
#include <stdio.h>
void foo2_1(void)
{
  printf("Test2 – foo1\n");
}
```

Library Archive

Creating a static library is as simple as creating an archive of object files using the ar command. If I have two source files named test1.c and test2.c, and I want to create a static library named libtest.a, then I would do the following:

$ arm-cortex_a8-linux-gnueabihf-gcc -c test1.c

$ arm-cortex_a8-linux-gnueabihf-gcc -c test2.c

$ arm-cortex_a8-linux-gnueabihf-ar rc libtest.a test1.o test2.o

Create a subdirectory called libs one directory above the current directory and copy the newly created static library libtest.a to it

Then to link libtest into my helloworld program, use:

$ arm-cortex_a8-linux-gnueabihf-gcc helloworld.c -ltest -L../libs -I../libs -o helloworld_S

$ arm-cortex_a8-linux-gnueabihf-readelf -a helloworld_S | grep "Shared library"

(make sure you are the correct directory where hellowworld.c is located)

Shared Libraries

The object code for a shared library must be position-independent, so that the runtime linker is free to locate it in memory at the next free address. To do this, add the -fPIC parameter to gcc, and then link it using the -shared option:

$ arm-cortex_a8-linux-gnueabihf-gcc -fPIC -c test1.c

$ arm-cortex_a8-linux-gnueabihf-gcc -fPIC -c test2.c

$ arm-cortex_a8-linux-gnueabihf-gcc -shared -o libtest.so test1.o test2.o

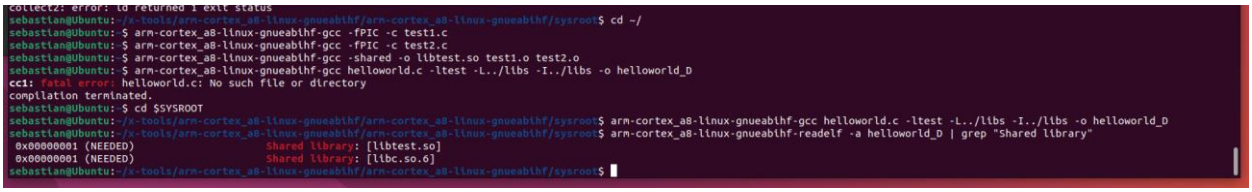
copy the newly created static library libtest.so to the directory libs created above

To link an application with this library, you add -ltest, exactly as in the static case mentioned in the preceding section, but this time the code is not included in the executable. Instead, there is a reference to the library that the runtime linker will have to resolve:

$ arm-cortex_a8-linux-gnueabihf-gcc helloworld.c -ltest -L../libs -I../libs -o helloworld_D

$ arm-cortex_a8-linux-gnueabihf-readelf -a helloworld_D | grep "Shared library"

As below:



If you want it to look for libraries in other directories as well, you can place a colon-separated list of paths in the shell variable LD_LIBRARY_PATH:

 # export LD_LIBRARY_PATH=/opt/lib:/opt/usr/lib

(Show screenshot of both created libraries for dynamic and static linking and describe the results of running the grep command on each of the files created dynamically and statically)

As below: