Writing and Compiling A Simple Program For OpenWrt

Filed under: Uncategorized — manoftoday @ 5:03 am

Writing and Compiling A Simple Program For OpenWrt

Written by Eric Bishop <ericpaulbishop-at-gmail.com>

Part I added 8/23/2007
Part II added 10/10/2007

Introduction

I’m writing this document because I found the documentation available on the OpenWrt wiki (http://wiki.openwrt.org) difficult to follow. I found myself in the position of wanting to compile a very simple program that I could run on my OpenWrt router. I ended up spending hours wading through the frustratingly incomplete documentation on the wiki, going through dozens of forum posts, and conducting extensive trial-and-error before my code would compile. I especially appreciated the examples on the wiki that contain the warning: “Note this Makefile is provided as an example only; it will not compile.” If something doesn’t work, it isn’t a very good example, is it? Here, then, is a (hopefully) more straightforward guide to building a program for OpenWrt. I found that existing documentation focuses more on porting existing, complicated programs to OpenWrt. My intention is to focus on getting a small, very simple, home-grown application running on OpenWrt. My goal is to explain this in as simple and complete a manner as possible, explaining each and every step necessary to write and compile a program that will run on OpenWrt. The process is actually very simple and straightforward — provided you know what you’re doing. For the purposes of this tutorial I’m going to assume you have a development box running linux and a router running OpenWrt. I will also assume you are at least somewhat familiar with C/C++ and standard Unix Makefiles.

The code for the examples in this tutorial can be downloaded from here. The example from the first part of the tutorial is in the openwrt-programming-examples/c directory and the example from the second part is in the openwrt-programming-examples/c++ directory.

Part I: A Simple Program in C

First, we’re going to need to write the code for the program itself and get it compiling on our local linux machine. Let’s write a simple “hello world” program that we want to run on the router:

~/helloworld/src/helloworld.c:

```
/****************
* Helloworld.c
* The most simplistic C program ever written.
* An epileptic monkey on crack could write this code.
*****************/
#include <stdio.h>

int main(void)
{
	printf("Hell! O' world, why won't my code compile?\n\n");
	return 0;
}
```

Alright, we have our code. Note the location of this file. Make a helloworld directory and then a src subdirectory. Place the code in the src subdirectory. Now, let’s write a standard Unix Makefile to compile this code for us:

~/helloworld/src/Makefile:

```
# build helloworld executable when user executes "make"
helloworld: helloworld.o
	$(CC) $(LDFLAGS) helloworld.o -o helloworld
helloworld.o: helloworld.c
	$(CC) $(CFLAGS) -c helloworld.c

# remove object files and executable when user executes "make clean"
clean:
	rm *.o helloworld
```

Notice that instead of hard-coding “gcc” in the makefile to compile the program, we use a variable that holds the C compiler, $(CC). If you’re compiling a c++ program you would use $(CXX) instead of $(CC) and $(CXXFLAGS) instead of $(CFLAGS). The use of the compiler variable is not necessary to compile the code locally, but in order to compile the code for OpenWRT it is critical because we won’t be using the standard gcc compiler. Place the makefile in the same src directory our code is in. We can now go to the src directory, type “make” and the program should compile. You can run it by typing “./helloworld”

```
mockingbird@linuxbox:~/helloworld/src$ make
cc -c helloworld.c
cc helloworld.o -o helloworld
mockingbird@linuxbox:~/helloworld/src$ ./helloworld
Hell! O' world, why won't my code compile?

mockingbird@linuxbox:~/helloworld/src$ make clean
rm *.o helloworld
mockingbird@linuxbox:~/helloworld/src$
```

So far this should be a review on how to write simple C programs and how to use Makefiles. Now comes the tricky part, compiling the code so that it will run on our router. The router uses a distinctly different architecture than our linux development box. Because there isn’t enough memory/disk space on the router to install a compiler and compile the code natively, we need to “cross-compile” the code on our development box for use on the router. To do this we need a special compiler and development environment called the OpenWRT SDK. You can download the SDK from http://downloads.openwrt.org The SDK varies depending on the architecture of your development box, the architecture of your router and the version/release of OpenWrt your router is running. I currently have whiterussian v0.9 installed on my Linksys WRT54G router, and my development box is an i686, so the SDK I use is this one. Extract the SDK files from the downloaded archive, and enter the SDK directory, which should have the same name as the tar.bz2 file (in my case OpenWrt-SDK-Linux-i686-1).

```
mockingbird@linuxbox:~$ tar xfj OpenWrt-SDK-Linux-i686-1.tar.bz2
mockingbird@linuxbox:~$ cd OpenWrt-SDK-Linux-i686-1
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ ls
dl  docs  examples  include  Makefile  package  README.SDK  rules.mk  scripts
staging_dir_mipsel
```

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$
```

Our goal is to build a package for OpenWrt using the source we already have. When you execute the “make” command in the SDK directory, the SDK will compile all properly configured packages in the package subdirectory under the SDK directory. The next step (and the trickiest) is to properly configure our code so that the SDK will build it. First, copy the helloworld directory we made earlier into the package subdirectory of the SDK:

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ cp -r ~/helloworld package
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$
```

In order to tell the OpenWrt SDK how to build our program we need to create a special Makefile in the helloworld directory, above the src directory which contains our conventional makefile. Writing this file is 90% of the work involved in compiling our program for OpenWrt. Below is an OpenWrt makefile for building the helloworld program. Each section is heavily commented so that it should be fairly clear what is going on:

~/OpenWrt-SDK-Linux-i686-1/package/helloworld/Makefile:

```
##############################################
# OpenWrt Makefile for helloworld program
#
#
# Most of the variables used here are defined in
# the include directives below. We just need to
# specify a basic description of the package,
# where to build our program, where to find
# the source files, and where to install the
# compiled program on the router.
#
# Be very careful of spacing in this file.
# Indents should be tabs, not spaces, and
# there should be no trailing whitespace in
# lines that are not commented.
#
##############################################

include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=helloworld
PKG_RELEASE:=1


# This specifies the directory where we're going to build the program.
# The root build directory, $(BUILD_DIR), is by default the build_mipsel
# directory in your OpenWrt SDK directory
PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)


include $(INCLUDE_DIR)/package.mk
```

```
# Specify package information for this program.
# The variables defined here should be self explanatory.
define Package/helloworld
	SECTION:=utils
	CATEGORY:=Utilities
	TITLE:=Helloworld -- prints a snarky message
	DESCRIPTION:=\
	If you can't figure out what this program does, \\\
	you're probably brain-dead and need immediate \\\
	medical attention.
endef


# Specify what needs to be done to prepare for building the package.
# In our case, we need to copy the source files to the build directory.
# This is NOT the default.  The default uses the PKG_SOURCE_URL and the
# PKG_SOURCE which is not defined here to download the source from the web.
# In order to just build a simple program that we have just written, it is
# much easier to do it this way.
define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef


# We do not need to define Build/Configure or Build/Compile directives
# The defaults are appropriate for compiling a simple program such as this one


# Specify where and how to install the program. Since we only have one file,
# the helloworld executable, install it by copying it to the /bin directory on
# the router. The $(1) variable represents the root directory on the router running
# OpenWrt. The $(INSTALL_DIR) variable contains a command to prepare the install
# directory if it does not already exist.  Likewise $(INSTALL_BIN) contains the
# command to copy the binary file from its current location (in our case the build
# directory) to the install directory.
define Package/helloworld/install
	$(INSTALL_DIR) $(1)/bin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/
endef


# This line executes the necessary commands to compile our program.
# The above define directives specify all the information needed, but this
# line calls BuildPackage which in turn actually uses this information to
# build a package.
$(eval $(call BuildPackage,helloworld))
```

As indicated, most OpenWrt make files specify how to download the source of an application from a URL, and most documentation assumes that you want to do this. However, if you’re building your own application from scratch it doesn’t make sense to download from a URL. It’s much simpler to have the source locally and use the Build/Prepare section to copy the source to the build directory, as shown above. Also, be very careful of spacing in the Makefile. The indentation under the define sections should be tabs, not spaces and there should be no whitespace at the end of lines that are not comments. The trailing whitespace can be a problem when variables are being defined, as the compiler may think there is a space at the end of a directory name. If we’re copying something to dir_with_trailing_space/subdir the copy command may be executed as “cp my.file dir_with_trailing_space /subdir”. Not only don’t you want anything in /subdir, you probably dont have permission to create it and write to it.

Now we’re all set to compile the helloworld package. Go to the root SDK directory (if you’re not already there) and type “make V=99” The “V=99” option is optional, but it is useful for debugging as it instructs the compiler to be “verbose” and output all the details of what it is doing.

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ make V=99
make package/compile
make[1]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1'
Collecting package info...
make -C package compile SDK=1
make[2]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[2]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[2]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make -j1 compile-targets
make[3]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make -C helloworld compile
make[4]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package/helloworld'
CFLAGS="-Os -pipe -mips32 -mtune=mips32 -funit-at-a-time -I/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/usr/include -I/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/include " LDFLAGS="-L/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/usr/lib -L/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/lib " make -C /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld AR=mipsel-linux-uclibc-ar AS="mipsel-linux-uclibc-gcc -c -Os -pipe -mips32 -mtune=mips32 -funit-at-a-time" LD=mipsel-linux-uclibc-ld NM=mipsel-linux-uclibc-nm CC="mipsel-linux-uclibc-gcc" GCC="mipsel-linux-uclibc-gcc" CXX=mipsel-linux-uclibc-g++ RANLIB=mipsel-linux-uclibc-ranlib STRIP=mipsel-linux-uclibc-strip OBJCOPY=mipsel-linux-uclibc-objcopy CROSS="mipsel-linux-uclibc-" CXXFLAGS="-Os -pipe -mips32 -mtune=mips32 -funit-at-a-time -I/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/usr/include -I/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/include " ARCH="mipsel" ;
make[5]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld'
make[5]: `helloworld' is up to date.
make[5]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld'
touch /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/.built
install -d -m0755 /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld/bin
install -m0755 /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/helloworld /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld/bin/
mkdir -p /home/mockingbird/OpenWrt-SDK-Linux-i686-1/bin/packages
find /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld -name CVS | xargs rm -rf
find /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld
-name .svn | xargs rm -rf
find /home/mockingbird/OpenWrt-SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld
-name '.#*' | xargs rm -f
STRIP="/home/mockingbird/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/bin/sstrip"
STRIP_KMOD="mipsel-linux-uclibc-strip --strip-unneeded --remove-section=.comment"
/home/mockingbird/OpenWrt-SDK-Linux-i686-1/scripts/rstrip.sh /home/mockingbird/OpenWrt-
SDK-Linux-i686-1/build_mipsel/helloworld/ipkg/helloworld
rstrip.sh: /home/mockingbird/OpenWrt-SDK-Linux-i686-
1/build_mipsel/helloworld/ipkg/helloworld/bin/helloworld:executable
ipkg-build -c -o 0 -g 0 /home/mockingbird/OpenWrt-SDK-Linux-i686-
1/build_mipsel/helloworld/ipkg/helloworld /home/mockingbird/OpenWrt-SDK-Linux-i686-
1/bin/packages
Packaged contents of /home/mockingbird/OpenWrt-SDK-Linux-i686-
1/build_mipsel/helloworld/ipkg/helloworld into /home/mockingbird/OpenWrt-SDK-Linux-i686-
1/bin/packages/helloworld_1_mipsel.ipk
make[4]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package/helloworld'
make[3]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[2]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[1]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1'
( \
        cd package; \
        find . -maxdepth 2 -name Config.in | \
                sed -e 's,/Config.in,,g' | \
                xargs -r -n1 make compile -C; \
)
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$
```

It compiled! The new package, helloworld_1_mipsel.ipk, is now located in the bin/packages subdirectory of the root SDK directory.

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ cd bin/packages
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1/bin/packages$ ls
helloworld_1_mipsel.ipk
mockingbird@linuxbox:~OpenWrt-SDK-Linux-i686-1/bin/packages$
```

This file is a ipk file which is used by the ipkg (itsy package management) system. Ipkg is a package management system for embedded devices, where space is an issue. Let’s copy this package onto the router, which is located at 192.168.1.1 on my network.

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1/bin/packages$ scp
helloworld_1_mipsel.ipk root@192.168.1.1:
root@192.168.1.1's password:
helloworld_1_mipsel.ipk                                   100% 1875     1.8KB/s   00:00
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1/bin/packages$
```

Now, ssh into the router. We just copied the package to root’s home directory so we are finally ready to install our program. In root’s home directory, (where we end up immediately after connecting to the router via ssh) type “ipkg install helloworld_1_mipsel.ipk” and the ipkg system will do the rest.

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1/bin/packages$ ssh root@192.168.1.1
root@192.168.1.1's password:
```

```
BusyBox v1.00 (2007.01.30-11:42+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 WHITE RUSSIAN (0.9) -------------------------------
  * 2 oz Vodka   Mix the Vodka and Kahlua together
  * 1 oz Kahlua  over ice, then float the cream or
  * 1/2oz cream  milk on the top.
 ---------------------------------------------------
root@OpenWrt:~# ls
TZ                     ip-up                  resolv.conf            spool
dhcp.leases            log                    resolv.conf.auto       usr
helloworld_1_mipsel.ipk  net                    run
root@OpenWrt:~# ipkg install helloworld_1_mipsel.ipk
Installing helloworld (1) to root...
Configuring helloworld
Successfully terminated.
root@OpenWrt:~#
```

The executable has now been installed into the /bin directory on the router, per our instructions in the OpenWrt Makefile listed above. So, all we have to do to run the program is type “helloworld” at the prompt. Note that because the executable has been installed to the /bin directory you should be able to execute the program no matter what directory you are in on the router.

```
root@OpenWrt:~# helloworld
Hell! O' world, why won't my code compile?

root@OpenWrt:~#
```

It works! Great success!

Now that you have a simple program compiling, you can start expanding on it to do whatever you want. If you run into problems I suggest you browse existing package Makefiles to see if someone else has done something similar to what you are trying to do. You can browse the files for these packages here. If you still have problems, browse the posts on the developer forum to see if anyone else has had a similar problem. If not, post your problem to the forum with a detailed explanation of what you’re trying to do, your OpenWrt Makefile and the errors you’re getting. Hopefully, some experienced developer will be kind enough to help out. Please do not email me personally if you have a problem with your code. If, however, you believe there is an error or serious omission in this tutorial, please let me know. I am relatively new to working with OpenWrt and it is certainly possible that I’ve made a mistake somewhere in this document. I have, however, personally tested all code included here on my own setup, and made every effort to be as accurate as possible.

Good luck with your programming!

Part II: C++ and the Standard Template Library (STL)

If you only want to compile C programs, the first part of my tutorial should be enough to get you started. However, if you want to use C++ there is another issue you're likely to run into. Let's say we want to compile the following C++ program for OpenWrt:

~/OpenWrt-SDK-Linux-i686-1/package/helloworld/src/helloworld.cpp:

```
/****************
* Helloworld.cpp
*****************/
#include <iostream>
#include <string>

using namespace std;

int main()
{
	string s = "Hell! O' world, why won't my c++ code run?\n\n";
	cout << s;
	return 0;
}
```

So, let's do exactly as we did before and see what happens. We just have to update the compiler flags in the Makefile to indicate that we're using a C++ compiler instead of a C compiler, right?

~/OpenWrt-SDK-Linux-i686-1/package/helloworld/src/Makefile:
NOTE: This code will compile but not run, further modifications are needed which are described below

```
# build helloworld executable when user executes "make"
helloworld: helloworld.o
	$(CXX) $(LDFLAGS) helloworld.o -o helloworld
helloworld.o: helloworld.cpp
	$(CXX) $(CXXFLAGS) -c helloworld.cpp

# remove object files and executable when user executes "make clean"
clean:
	rm *.o helloworld
```

If you use the exact same OpenWrt Makefile as before, this program will compile just fine. It will even install properly when you copy the helloworld_1_mipsel.ipk file to the router and type "ipkg install helloworld_1_mipsel.ipk" However, watch what happens if you try to run the program:

```
root@OpenWrt:~# helloworld
helloworld: can't load library 'libstdc++.so.6'
root@OpenWrt:~#
```

So, what went wrong? The problem is that this program uses strings and iostreams which are a feature of the C++ standard template library (STL). However, because memory is so critical in an embedded application like OpenWrt, the standard template library is not available. Instead, we need to link to a special implementation of the standard library for embedded devices called uClibc++. This library implements the same functions and data structures as the standard library but takes up less memory.

Before we adjust our Makefiles to use to this alternate library, let’s first make sure that it’s installed on the router so that it is available to link to. Log into your router. Then, make sure your list of packages is up to date by using the “ipkg update” command:

```
root@OpenWrt:~# ipkg update
Downloading http://downloads.openwrt.org/backports/0.9/Packages
Updated list of available packages in /usr/lib/ipkg/lists/0.9-backports
Downloading http://download2.berlios.de/pub/xwrt/packages/Packages
Updated list of available packages in /usr/lib/ipkg/lists/X-Wrt
Downloading http://downloads.openwrt.org/whiterussian/packages/Packages
Updated list of available packages in /usr/lib/ipkg/lists/whiterussian
Downloading http://downloads.openwrt.org/whiterussian/packages/non-free/Packages
Updated list of available packages in /usr/lib/ipkg/lists/non-free
Successfully terminated.
root@OpenWrt:~#
```

Once you have updated your package list, install uclibc++ by typing “ipkg install uclibc++” If you do not have uclibc++ installed this should install it:

```
root@OpenWrt:~# ipkg install uclibc++
Installing uclibc++ (0.1.11-2) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/uclibc++_0.1.11-2_mipsel.ipk
Configuring uclibc++
Successfully terminated.
root@OpenWrt:~#
```

Otherwise, if the library is already installed, you’ll see this confirmation:

```
root@OpenWrt:~# ipkg install uclibc++
Package uclibc++ (0.1.11-2) installed in root is up to date.
Nothing to be done
Successfully terminated.
root@OpenWrt:~#
```

Now it’s time to modify the Makefiles. First, the one in the src directory. Only a small change is necessary here, to the line where we link the objects together. We need to be able to tell the linker to link to another library, and the way we do that is by defining a variable called $(LIBS). The variable will be defined in the special OpenWrt Makefile, but will be used here, so we need to add it to the end of the line which specifies how to do the linking:

~/OpenWrt-SDK-Linux-i686-1/package/helloworld/src/Makefile:

```
# build helloworld executable when user executes "make"
helloworld: helloworld.o
	$(CXX) $(LDFLAGS) helloworld.o -o helloworld $(LIBS)
helloworld.o: helloworld.cpp
	$(CXX) $(CXXFLAGS) -c helloworld.cpp

# remove object files and executable when user executes "make clean"
clean:
	rm *.o helloworld
```

The OpenWrt Makefile needs a slightly more complex modification. Recall the comment I placed in the OpenWrt Makefile we used before, “We do not need to define Build/Configure or Build/Compile directives. The defaults are appropriate for compiling a simple program such as this one.” Well, guess what? In order to link to uclibc++ we need to customize how we’re going to compile the program, and we’re going to do this in the Build/Compile directive. Before we re-define this directive to suit our purposes, let’s see what the default looks like:

TAKEN FROM: ~/OpenWrt-SDK-Linux-i686-1/include/package.mk:

```
define Build/Compile/Default
	CFLAGS="$(TARGET_CFLAGS) $(EXTRA_CPPFLAGS) " \
	LDFLAGS="$(EXTRA_LDFLAGS) " \
	$(MAKE) -C $(PKG_BUILD_DIR) \
		$(TARGET_CONFIGURE_OPTS) \
		CROSS="$(TARGET_CROSS)" \
		CXXFLAGS="$(TARGET_CFLAGS) $(EXTRA_CPPFLAGS) " \
		ARCH="$(ARCH)" \
		$(1);
endef
```

This code specifies the make command (the third line) and a bunch of flags and parameters for compilation. In order to link to uClibc++ we add the definition of the $(LIBS) variable as “-nodefaultlibs -lgcc -lc -luClibc++” Also, we need to specify “-nostdinc++” in the compiler flags to tell the compiler that c++ standard template library functions and data structures will be linked to in specified external libraries and not the standard libraries:

~/OpenWrt-SDK-Linux-i686-1/package/helloworld/Makefile:

```
##############################################
# OpenWrt Makefile for helloworld program
#
#
# Most of the variables used here are defined in
# the include directives below. We just need to
# specify a basic description of the package,
# where to build our program, where to find
# the source files, and where to install the
# compiled program on the router.
#
# Be very careful of spacing in this file.
# Indents should be tabs, not spaces, and
# there should be no trailing whitespace in
# lines that are not commented.
#
##############################################

include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=helloworld
```

```
PKG_RELEASE:=1


# This specifies the directory where we're going to build the program.
# The root build directory, $(BUILD_DIR), is by default the build_mipsel
# directory in your OpenWrt SDK directory
PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)


include $(INCLUDE_DIR)/package.mk



# Specify package information for this program.
# The variables defined here should be self explanatory.
define Package/helloworld
	SECTION:=utils
	CATEGORY:=Utilities
	TITLE:=Helloworld -- prints a snarky message
	DESCRIPTION:=\
	If you can't figure out what this program does, \\\
	you're probably brain-dead and need immediate \\\
	medical attention.
endef


# Specify what needs to be done to prepare for building the package.
# In our case, we need to copy the source files to the build directory.
# This is NOT the default.  The default uses the PKG_SOURCE_URL and the
# PKG_SOURCE which is not defined here to download the source from the web.
# In order to just build a simple program that we have just written, it is
# much easier to do it this way.
define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef



################################################################################
#########
# The Build/Compile directive needs to be specified in order to customize compilation
# and linking of our program.  We need to link to uClibc++ and to specify that we
# do NOT want to link to the standard template library.
#
# To do this we define the LIBS variable.  To prevent linking to the standard libraries we
# add "-nodefaultlibs" to the $(LIBS) variable and then specify "-lgcc -lc" to ensure that
# there are no unresolved references to internal GCC library subroutines. Finally
# "-luClibc++" to link to the  uClibc++ library.  Also, we need to specify "-nostdinc++"
# in the compiler flags to tell the compiler that c++ standard template library functions
# and data structures will be linked to in specified external libraries and not the
# standard libraries.
```

```
##################################################################################
#########
define Build/Compile
	$(MAKE) -C $(PKG_BUILD_DIR) \
		LIBS="-nodefaultlibs -lgcc -lc -luClibc++" \
		LDFLAGS="$(EXTRA_LDFLAGS)" \
		CXXFLAGS="$(TARGET_CFLAGS) $(EXTRA_CPPFLAGS) -nostdinc++" \
		$(TARGET_CONFIGURE_OPTS) \
		CROSS="$(TARGET_CROSS)" \
		ARCH="$(ARCH)" \
		$(1);
endef


# Specify where and how to install the program. Since we only have one file,
# the helloworld executable, install it by copying it to the /bin directory on
# the router. The $(1) variable represents the root directory on the router running
# OpenWrt. The $(INSTALL_DIR) variable contains a command to prepare the install
# directory if it does not already exist.  Likewise $(INSTALL_BIN) contains the
# command to copy the binary file from its current location (in our case the build
# directory) to the install directory.
define Package/helloworld/install
	$(INSTALL_DIR) $(1)/bin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/
endef


# This line executes the necessary commands to compile our program.
# The above define directives specify all the information needed, but this
# line calls BuildPackage which in turn actually uses this information to
# build a package.
$(eval $(call BuildPackage,helloworld))
```

That’s it! That modification should be sufficient to enable linking to uClibc++. Compile it as before, copy the .ipk file to the router, install it with ipkg and run it:

```
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ make clean
rm -rf build_* bin
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ make
make package/compile
make[1]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1'
Collecting package info...
make -C package compile SDK=1
make[2]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[2]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[2]: Entering directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[4] -C package compile-targets
make[5] -C package/helloworld compile
make[2]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1/package'
make[1]: Leaving directory `/home/mockingbird/OpenWrt-SDK-Linux-i686-1'
( \
      cd package; \
        find . -maxdepth 2 -name Config.in | \
                sed -e 's,/Config.in,,g' | \
                xargs -r -n1 make compile -C; \
)
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ scp
bin/packages/helloworld_1_mipsel.ipk root@192.168.1.1:
root@192.168.1.1's password:
helloworld_1_mipsel.ipk                          100% 2570     2.5KB/s   00:00
mockingbird@linuxbox:~/OpenWrt-SDK-Linux-i686-1$ ssh root@192.168.1.1
root@192.168.1.1's password:


BusyBox v1.00 (2007.01.30-11:42+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 WHITE RUSSIAN (0.9) -------------------------------
  * 2 oz Vodka   Mix the Vodka and Kahlua together
  * 1 oz Kahlua  over ice, then float the cream or
  * 1/2oz cream  milk on the top.
 ---------------------------------------------------
root@OpenWrt:~# ipkg install helloworld_1_mipsel.ipk
Installing helloworld (1) to root...
Configuring helloworld
Successfully terminated.
root@OpenWrt:~# helloworld
Hell!  Why won't my c++ code run?
root@OpenWrt:~#
```

It works! Our Jedi programming skills have prevailed!

Once again, please do not contact me about any issues you might have with your own programs. If, however, you think there is an error in one of my examples or that I have omitted a critical detail, please contact me and I will try to resolve the issue as soon as possible.