Cross-Compiling: Redis as middleware

Release

Vincent Gijsen

Sep 13, 2017

CONTENTS

Table 1: Version information

Version	Remarks
0.1	Initial version, containing redis-server compilation
0.2	Altered structure and added CPP redis client code
0.3	Fixed typo in *gnuabi-cmd* and PREFIX variable in *cpp-redis* build steps

CHAPTER

ONE

SETUPCROSSCOMPILER

We need to tell where we would like our compiled libraries to be 'installed' so we create `/opt/pxc/setupCrosscompiler` this way we dont taint our host with ARM binaries.

```
export ARMPREFIX=/opt/pxc/install
export PKG_CONFIG_PATH=$ARMPREFIX/lib/pkgconfig:$PKG_CONFIG_PATH
```

we can make these settings active in the current session by exeucting

```
vincent@vincent-VirtualBox ~ $ source /opt/pxc/setupCrossCompiler
```

CHAPTER

TWO

TOOLCHAIN.CMAKE

We also need to create a `toolchain.cmake` file.

Create the file somewhere like `/opt/pxc/toolchain.cmake`

Its content:

```
#!/bin/sh

set(ENV{PKG_CONFIG_PATH} $ENV{ARMPREFIX}/lib/pkgconfig)
set(ENV{LD_LIBRARY_PATH} $ENV{ARMPREFIX}/lib)
set(ENV{C_INCLUDE_PATH} $ENV{ARMPREFIX}/include)
set(ENV{CPLUS_INCLUDE_PATH} $ENV{ARMPREFIX}/include)

set(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} $ENV{ARMPREFIX}/lib)
set(MY_FLAGS "-I$ENV{ARMPREFIX}/include -L$ENV{ARMPREFIX}/lib")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}  ${MY_FLAGS}")
set(CMAKE_C_FLAGS   "${CMAKE_C_FLAGS}  ${MY_FLAGS}")

set(CMAKE_INSTALL_PREFIX $ENV{ARMPREFIX})
```

We will need to reference to the toolchain file for `CMake projects`

to do so ,add it a s parameter, e.g.

```
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/pxc/toolchain.cmake  ../
```

This ensures the PxC Compiler is selected, and the installation directory is set according to `$ARMPREFIX`.

CHAPTER

THREE

COMPILING REDIS SERVER

Warning: Note there are many ways to accomplish this (and perhaps better ones!), this is just one way of doing things...

In this example, we are using Redis to share data between the real-time and non-realtime system. The goal is to use our vision-application, and share information between the vision app- and the PLC-logic

3.1 Assumptions

In order to follow this manual keep in mind:

- I use a Linux-workstation (or Virtual Machine)
- The PxC Linux-sdk is installed in */opt/pxc/current*
- Our working dir is */opt/pxc*
- Our installation dir where we keep the ARM cross-compild libraries is */opt/pxc/current*
- We used Redis 4.0.1 stable at the time of this tutorial

3.2 Building the Redis Server application

For starters, we are going to cross-compile a Redis-database server, so it runs on the PLCNext non-realtime linux environment.

First we need to get the redis server from https://redis.io source-code to our development system

Execute below on your Linux system (not the plc):

```
cd /opt/pxc/
wget http://download.redis.io/releases/redis-4.0.1.tar.gz
tar -xzf redis-4.0.1.tar.gz
cd redis-4.0.1
```

Now we are looking at Redis source code.

we need to have run the environment setup command:

```
source /opt/pxc/current/environment-setup-cortexa9t2hf-neon-pxc-linux-
↪gnueab
```

After we run the command *make* we will see some errors:

```
MAKE hiredis
cd hiredis && make static
make[3]: Entering directory '/opt/pxc/redis-4.0.1/deps/hiredis'
gcc -std=c99 -pedantic -c -O3 -fPIC -O2 -pipe -g -feliminate-unused-debug-
↪types -Wall -W -Wstrict-prototypes -Wwrite-strings -g -ggdb arm net.c
gcc: error: arm: No such file or directory
Makefile:156: recipe for target 'net.o' failed
make[3]: *** [net.o] Error 1
make[3]: Leaving directory '/opt/pxc/redis-4.0.1/deps/hiredis'
Makefile:45: recipe for target 'hiredis' failed
make[2]: *** [hiredis] Error 2
make[2]: Leaving directory '/opt/pxc/redis-4.0.1/deps'
Makefile:170: recipe for target 'persist-settings' failed
make[1]: [persist-settings] Error 2 (ignored)
    CC adlist.o
In file included from adlist.c:34:0:
zmalloc.h:50:31: fatal error: jemalloc/jemalloc.h: No such file or␣
↪directory
#include <jemalloc/jemalloc.h>
                              ^
compilation terminated.
Makefile:228: recipe for target 'adlist.o' failed
make[1]: *** [adlist.o] Error 1
make[1]: Leaving directory '/opt/pxc/redis-4.0.1/src'
Makefile:6: recipe for target 'all' failed
make: *** [all] Error 2
```

We might fix this, as we found that ther *ARCH* variable causes the error, as it seems to break things, we make it empty:

```
make ARCH=
```

This time, we it does work more consistently, but still an error. Studying the readme tels that we need to tell the make-file which memory-allocator implementation we would like to use. To ensure we don't try to link

```
make ARCH= MALLOC=libc
```

Unfortunatly this also results in an error, as the Makefile and the PxC environment don't play nice. This is due the way the makefile checks wether the *$CPP/$CC* variables and how they are populated

the exact line in *deps/hires/Makefile*:

```
38 # Fallback to gcc when $CC is not in $PATH.
39 CC:=$(shell sh -c 'type $(CC) >/dev/null 2>/dev/null && echo $(CC) ||␣
↪echo gcc')
40 CXX:=$(shell sh -c 'type $(CXX) >/dev/null 2>/dev/null && echo $(CXX)␣
↪|| echo g++')
41 OPTIMIZATION?=-O3
42 WARNINGS=-Wall -W -Wstrict-prototypes -Wwrite-strings
```

By commenting out these two lines by adding a *#* in front of the *CC:* and *CXX:* and running the below commands, we now build this dependency with the cross-compiler, instead of the system's compiler (which compiles to x86). After which the linker can also do his job:

```
...
34 unixsocket /tmp/hiredis-test-redis.sock
35 endef
36 export REDIS_TEST_CONFIG
37
38 # Fallback to gcc when $CC is not in $PATH.
39 #CC:=$(shell sh -c 'type $(CC) >/dev/null 2>/dev/null && echo $(CC) ||␣
↪echo gcc')
40 #CXX:=$(shell sh -c 'type $(CXX) >/dev/null 2>/dev/null && echo $(CXX)␣
↪|| echo g++')
41 OPTIMIZATION?=-O3
42 WARNINGS=-Wall -W -Wstrict-prototypes -Wwrite-strings
43 DEBUG_FLAGS?= -g -ggdb
44 REAL_CFLAGS=$(OPTIMIZATION) -fPIC $(CFLAGS) $(WARNINGS) $(DEBUG_FLAGS)
↪$(ARCH)
45 REAL_LDFLAGS=$(LDFLAGS) $(ARCH)
```

and executing:

```
make distclean
make ARCH= MALLOC=libc
```

Now it compiles beautifully :)

```
    ...
    LINK redis-server
    INSTALL redis-sentinel
    CC redis-cli.o
    LINK redis-cli
    CC redis-benchmark.o
    LINK redis-benchmark
    INSTALL redis-check-rdb
    INSTALL redis-check-aof

Hint: It's a good idea to run 'make test' ;)

make[1]: Leaving directory '/opt/pxc/redis-4.0.1/src'
```

we are now ready to install the files in our local ARM-compiled binaries, but we need a small addition to the regular command.

```
make PREFIX=$ARMPREFIX install
```

This installs in our *$ARMPREFIX* directory

```
vincent@vincent-VirtualBox /opt/pxc/redis-4.0.1 $ make PREFIX=$ARMPREFIX␣
↪install
cd src && make install
make[1]: Entering directory '/opt/pxc/redis-4.0.1/src'

Hint: It's a good idea to run 'make test' ;)

    INSTALL install
    INSTALL install
    INSTALL install
    INSTALL install
```

```
    INSTALL install
make[1]: Leaving directory '/opt/pxc/redis-4.0.1/src'
```

Now you might copy from */opt/pxc/intall*:

- bin/redis-benchmark
- bin/redis-check.aof
- bin/redis-check-rdb
- bin/redis-cli
- bin/redis-server

To the PLCNext controller, and use them.

3.3 Post Compilation

Having the binaries on the controller, is nice, makeing it autostart is even nicer :)

So we first copy from the source directory the file *utils/redis_init_script* to the controller at location */etc/init.d/* and make it executeble with *chmod +x <name of file>*

In this file, we need to set a few properties:

```
REDISPORT=6379
EXEC=/usr/local/bin/redis-server
CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis_${REDISPORT}.pid
CONF="/etc/redis/${REDISPORT}.conf"
```

In our case we copied the files to /home/root/redis.

So change the paths *EXEC* and *CLIEXEC*.

Warning: Ensure that our redis-server is running in daemon node, by setting the configuration-file property *daemonize yes*. Otherwise you block other services from starting, such as the PLCRuntime!

Do note we also need to copy a valid configuration file, in the designated path, in this example /etc/redis/6379.conf. But I suggest keeping things simple like */home/root/redis/redis.conf*.

The final step is to create a symlink in */etc/rc5.d/* this needs to have a name like *S100redis* which ensures our redis server is started late in this runlevel.

```
ln -s /etc/init.d/redis_init_script /etc/rc5.d/S100redis
```

CHAPTER

FOUR

COMPILING A REDIS CLIENT

We need to get a client to insert into the Realtime C++, but redis has serveral, just for c++ there are redis, so we need to set some Selection criterias:

Selection criterials:

- Licence type
- Dependencies
- Light weight
- 'simple'
- thread safety
- need for connection pooling

As we are browsing the various C++ clients. We decide that the *cpp-redis* meets our demands. It is available via https://github.com/Cylix/cpp_redis. Getting the source is easy when it is hosted on *Github*.

Press the green button on the webpage to get the *git* url.

By issueing *git clone* <*path*> we get the latest and greatest.

```
git clone https://github.com/Cylix/cpp_redis
```

Tip: Should you don't have *git* installed, install it via *sudo apt-get install git* when on Debian/Ubuntu based systems.

We now have the cpp-redis application's source-code. Which needs to be compile in order to use in our own application later on.

4.1 Compiling cpp-redis

Go into the *cpp-redis* direcotry

```
cd cpp_redis
```

We spot that this might have a *CMake* build system.

If you don't have CMake tools available.

```
sudo apt-get install cmake
```

reading the wiki on this particular c++ library (https://github.com/Cylix/cpp_redis/wiki) tells us, that we need to take a few steps:

```
# Clone the project
git clone https://github.com/Cylix/cpp_redis.git
# Go inside the the project directory
cd cpp_redis
# Get tacopie submodule
git submodule init && git submodule update
# Create a build directory and move into it
mkdir build && cd build
# Generate the Makefile using CMake
cmake .. -DCMAKE_BUILD_TYPE=Release
# Build the library
make
# Install the library
make install
```

If we would blindly execute these functions, we would end up with a nice *Intel x86* binary, not very usefull on a ARM platform. But we need some to configure our crosscompile environment prior to the execution of *cmake* command and *make* commands.

so first we start with:

```
# Clone the project
git clone https://github.com/Cylix/cpp_redis.git
# Go inside the the project directory
cd cpp_redis
# Get tacopie submodule
git submodule init && git submodule update
# Create a build directory and move into it
mkdir build && cd build
```

than we ensure our environment is configured correctly, so execute (if not already done so):

```
source /opt/pxc/current/environment-setup-cortexa9t2hf-neon-pxc-linux-
↪gnueabi
source /opt/pxc/setupCrossCompiler
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_TOOLCHAIN_FILE=/opt/pxc/
↪toolchain.cmake -DCMAKE_INSTALL_PREFIX=$ARMPREFIX
```

This genertes the `Makefile` and auxilary files needed for the actual compilation:

```
make
make install
```

If all is well, the library is compiled and installed into the `$ARMPREFIX`