

Rock Rtems

Ana Vázquez Alonso

Índice General

Before starting

On this tutorial you will find the necessary steps to build Rock on top of RTEMS using x86 or SPARC as targets. The aim of this tutorial is not to explain how RTEMS or Rock work, for more information, please visit the projects website.

These patches and scripts have been developed and tested in Ubuntu 10.04.2 LTS with GCC 4.4.3, Ubuntu 10.10 with GCC 4.5.1 and in a Gentoo system with GCC 4.5.2, Binutils 2.21 and GLIBC 2.13-r2. Also notice that at this point, these scripts use specific bash features (such as parameter expansions, pattern substitution, etc), which make bash use mandatory.

Before starting the installation, you need to download the automatic build scripts (from http://roble.datsi.fi.upm.es/~anita/rock/scripts_05.tar.bz2) and create the following directory structure:

```
mkdir -p rock/buildrock/build/pkgsrock/install
cd rock / build
wget http://roble.datsi.fi.upm.es/~anita/rock/scripts_05.tar.bz2
tar xvjf scripts_05.tar.bz2
```

All the scripts and patches needed to compile Rock on top of RTEMS should be located in rock/build/scripts/
You need to tailor the env.sh script (in rock/build/scripts/env.sh) in order to set the path and the target:

```
export BUILD_PREFIX = full path to the build folder you have just created : rock / build
export INSTALL_PREFIX = full path to the installation directory you have just created : rock / install
```

```
# ERC32 target
# export TARGET = sparc - rtems
# export BSP = erc32
# export ARCH = sparc
```

```
# LEON2 target
# export TARGET = sparc - rtems
# export BSP = leon2
# export ARCH = sparc
```

```
# LEON3 target
# export TARGET = sparc - rtems
# export BSP = leon3
# export ARCH = sparc
```

```
# x86 target
export TARGET = i386 - rtems
export BSP = pc486
export ARCH = x86
```

Now issue the commands below:

```
cd $BUILD_PREFIX/
source scripts/env . sh
```

Rtems

The first step for compile RTEMS is setting up all the cross compiler tools for x86/SPARC. For these purposes we use the following packages

- Binutils: version 2.20.1
- Newlib: version 1.18.0
- GCC: version 4.5.2, C and C++ support
- RTEMS: version 4.10.1

In order to simplify this task, an automatic script has been written. It will download automatically all the necessary packages and will proceed with the installation based on the selected target (in rock/build/scripts/env.sh).

To accomplish the next step, you need to be sure that BUILD_PREFIX and INSTALL_PREFIX are properly set (as explained in chapter 1).

```
cd $BUILD_PREFIX/  
./scripts/prepare_rtems.sh
```

The prepare rtems.sh script will download all the needed packages to \$BUILD_PREFIX/pkgs/ (if they are not already there), uncompress them and apply the patches.

Now you can proceed to compile the packages:

```
cd $BUILD_PREFIX/rtems  
make build-binutils && make build-gcc && make build-gdb && make build-rtems ( or make all )
```

. Testing the installation

You can test your RTEMS instalation with the RTEMS 4.10.1 examples <http://www.rtems.org/ftp/pub/rtems/4.10.1/examples-v2-4.10.1.tar.bz2>.

Boost

Boost provides free peer-reviewed portable C++ source libraries.

. Compiling Boost

Issue the commands below in order to download, uncompress, patch and compile Boost 1.44.0:

```
cd $BUILD_PREFIX  
./scripts/boost_cross_compiler.sh
```

After completion of the compilation process, if everything went well, you should have a boost folder in your \$INSTALL_PREFIX with the libraries below:

- libboost filesystem.a
- libboost regex.a
- libboost serialization.a
- libboost system.a
- libboost thread.a
- libboost wserialization.a
- libboost graph.a
- libboost unit test framework.a
- libboost test exec monitor.a
- libboost prg exec monitor.a

and some .h files.

omniORB

omniORB is a robust high performance CORBA ORB for C++ and Python. It is freely available under the terms of the GNU Lesser General Public License (for the libraries), and GNU General Public License (for the tools). omniORB is largely CORBA 2.6 compliant.

. Compiling omniORB

Issue the commands below in order to download, uncompress, patch and compile omniORB 4.1.4:

```
cd $BUILD_PREFIX  
./scripts/omniORB_cross_compiler.sh
```

After completion of the compilation process, if everything went well, you should have a omniORB folder in your \$INSTALL_PREFIX with the needed libraries, include files and binaries.

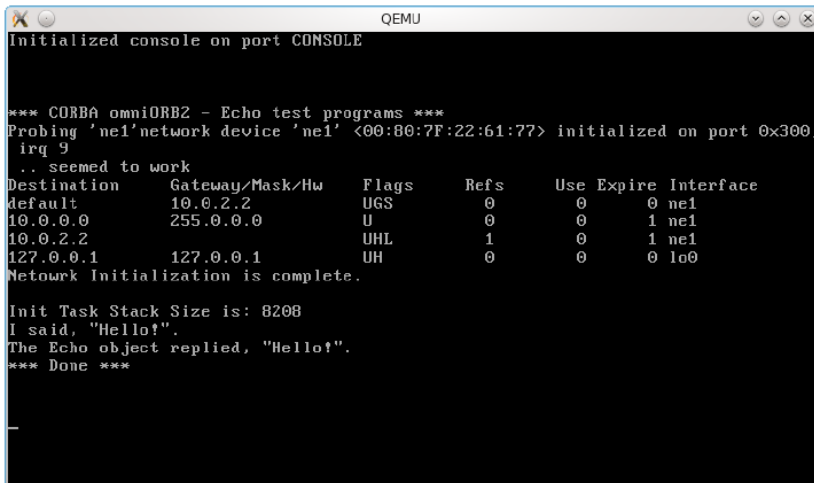
. Testing omniORB

In order to test your omniORB installation you can use the omiORB examples together with the configuration files included in omniORB rtemsExamples.tar.bz2:

```
cd $BUILD_PREFIX/omniORB-4.1.4/src/examples  
tar xvjf $BUILD_PREFIX/scripts/files/omniORB_rtemsExamples.tar.bz2  
cd echo(or any other example)  
make
```

Notice that to run these tests you need TCP/IP support in QEMU (see section 8.2).

Figura 1: omniORB echo test



```
QEMU
Initialized console on port CONSOLE

*** CORBA omniORB2 - Echo test programs ***
Probing 'ne1' network device 'ne1' <00:80:7F:22:61:77> initialized on port 0x300,
irq 9
.. seemed to work
Destination      Gateway/Mask/Hw  Flags    Refs    Use Expire Interface
default          10.0.2.2         UGS      0       0       0 ne1
10.0.0.0         255.0.0.0        U        0       0       1 ne1
10.0.2.2         10.0.2.2         UHL      1       0       1 ne1
127.0.0.1        127.0.0.1        UH       0       0       0 lo0
Network Initialization is complete.

Init Task Stack Size is: 8208
I said, "Hello!".
The Echo object replied, "Hello!".
*** Done ***
```

Eigen

. Compiling Eigen

Issue the commands below in order to download, uncompress and compile Eigen 2.0.16:

```
cd $BUILD_PREFIX
./scripts/eigen_cross_compiler.sh
```

After completion of the compilation process, if everything went well, you should have a eigen folder in your \$INSTALL_PREFIX with the needed .h files.

LibXML2

libxml2 is a software library for parsing XML documents. It is also the basis for the libxslt library which process XSLT-1.0 stylesheets.

. Compiling LibXML

Issue the commands below in order to download, uncompress and compile LibXML2:

```
cd $BUILD_PREFIX
./scripts/libxml.sh
```

After completion of the compilation process, if everything went well, you should have a libxml2 folder in your \$INSTALL_PREFIX with the needed library files.

Xerces

Xerces-C++ is a validating XML parser written in a portable subset of C++ that makes easy to give your application the ability to read and write XML data.

. Compiling Xerces

Issue the commands below in order to download, uncompress, patch and compile Xerces-C++ 3.1.1:

```
cd $BUILD_PREFIX
./scripts/xerces_c_cross_compiler.sh
```

After completion of the compilation process, if everything went well, you should have a xerces folder in your \$INSTALL_PREFIX with the needed libraries, include files and binaries.

Rock

. Compiling Rock

Since Rock is in continuous development, the patches for Rock have been developed based on a stable frozen Git version. A tar package with this version can be downloaded from http://roble.datsi.fi.upm.es/~anita/rock/rock_base.tar.bz2.

```
cd $BUILD_PREFIX/pkgs
wget http://roble.datsi.fi.upm.es/~anita/rock/rock_base.tar.bz2
```

Then execute the commands below in order to uncompress, compile (using autoproj build) utilmm and typelib for the host system, and finally patch Rock:

```
cd $BUILD_PREFIX
./scripts/prepare_rock.sh
```

Compile Rock by issuing the following commands (it will automatically compile target packages using the cross compiler):

```
cd $BUILD_PREFIX/rock
source env . sh
autoproj build
This is the output generated if everything goes well:
$ autoproj build
  Access method to import data from gitorious . org ( git , http or ssh ) : git
  Which prepackaged software ( a.k.a. ' osdeps ' ) should autoproj install automatically ( all , ruby , os , none ) ? all
autoproj : loading ...
run ' autoproj reconfigure ' to change configuration options and use ' autoproj switch-config ' to change the remote source for autoproj ' s main build configuration
  Which flavor of Rock do you want to use ' next the target operating system for Orocos/RTT ( gnulinux , xenomai or rtems ) : rtems
  which CORBA implementation should the RTT use ? omniORB
  Do you need compatibility with OCL ? ( yes or no ) : false

autoproj : importing and loading selected packages
  WARN : base/templates/vizkit from rock . base does not have a manifest
  WARN : external/sisl from rock . base does not have a manifest
autoproj : building and installing packages
  configuring CMake build system for utilmm
  building utilmm (100%)
  installing utilmm
  building external/sisl
  installing external/sisl
  configuring CMake build system for base/types
  building base/types (100%)
  installing base/types
  configuring CMake build system for rtt
  building rtt (100%)
  installing rtt
  setting up Ruby package utilrb
  configuring CMake build system for typelib
  building typelib (100%)
  installing typelib
  setting up Ruby package orogen
  setting up Ruby package base/types_ruby
  generating oroGen project base/orogen/types
  configuring CMake build system for base/orogen/types
  building base/orogen/types (100%)
  installing base/orogen/types
  setting up Ruby package tools/pocolog
  setting up Ruby package tools/roby
  setting up Ruby package base/scripts

autodetected the shell to be bash , sourcing autoproj shell helpers
add " Autoproj . shell_helpers = false " in autoproj/init . rb to disable
autoproj : updated/home/anita/rock/build/rock/env . sh
Build finished successfully at Sun Oct 09 00:43:48 +0200 2011
```

. Compile Rock test

Libraries and binaries can be built using the commands below:

```
mkdir $BUILD_PREFIX/rock/tutorial && cd $BUILD_PREFIX/rock/tutorial
git clone git://gitorious.org/rock-tutorials/message_driver.git
cp -r $BUILD_PREFIX/rock/base/templates/cmake_lib/cmake message_driver/
( this step is needed since this library has been already generated .
If you use rock - create - lib in order to create a new library , this
directory will be copied automatically if $OROCOS_TARGET == rtems )
autoproj build message-driver
mkdir $BUILD_PREFIX/rock/tutorial/orogen && cd $BUILD_PREFIX/rock/tutorial/orogen
git clone git://gitorious.org/rock-tutorials/orogen-message_producer.git
autoproj build message-producer
```

Some minor adaptations are needed since the rock base used (from http://roble.datsi.fi.upm.es/~anita/rock/rock_base.tar.bz2) is not as updated as git tutorials (TaskBase(name, engine, initial state) to TaskBase(name, initial state), msg.time.toString() to msg.time).

. Testing periodic activities on Rock

You can test your instalation running the start.rb script on the host machine. In this case you will need to comment the run action since the deployer should be manually started with Qemu.

http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/ruby_start.ogv^[1]

You can also connect your components specifying in rtems_init.cpp your ip configuration.

http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/message_consumer.ogv^[2]

http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/message_manager.ogv^[3]

Using QEMU for RTEMS PC (i386) BSP

QEMU is a generic machine emulator that can be used to run RTEMS PC (i386) BSP. Although this guide will walk you through the basic QEMU configuration to run Rock applications, you should also take some time to read the RTEMS documentation for QEMU: <http://www.rtems.com/wiki/index.php/QEMU>.

You can run a Rock application with QEMU by typing:

```
qemu -kernel application.exe
```

Normally, QEMU uses SDL to display the VGA output. With -curses, QEMU can display the VGA output when in text mode using a curses/ncurses interface.

. GDB usage

QEMU has a primitive support to work with gdb, so that you can do ctrl-c while the virtual machine is running and inspect its state:

```
qemu -kernel application.exe -s -S
```

Then launch gdb on the application.exe executable, and connect to QEMU:

```
gdb application . exe
```

```
(gdb) target remote localhost :1234
```

Then you can use gdb normally. For example, type c to launch the kernel:

```
(gdb) c
```

. Network emulation

In order to use omniNames you need to enable network support in QEMU. QEMU can simulate a network card and can connect it to an arbitrary number of Virtual Local Area Networks (VLANs). QEMU adds a virtual network device on your host (called tapN), and you can then configure it as if it was a real ethernet card.

.. Configure the Linux kernel for bridge options

You must enable TUN/TAP in your kernel by using the following configuration option:

Linux Kernel Configuration: Enable TUN/TAP support

```
Device Drivers --->
```

```
Networking support --->
```

```
<M> Universal TUN/TAP device driver support
```

```
Networking --->
```

```
Networking options --->
```

```
<M> 802.1d Ethernet Bridging #NOTE : at least for 2.6.20 series
```

Load the tun module:

```
modprobe tun
```

.. Configure a network bridge in Gentoo

First, install the necessary packages for the bridge management:

```
emerge net-misc/bridge-utils sys-apps/usermode-utilities
```

Create the bridge:

```
sudo brctl addbr br0
```

Specify the IP used for the bridge in /etc/conf.d/net

Using a static IP configuration:

```
bridge_br0="eth0"
config_eth0=( "null" )
config_br0=( "192.168.0.1/24" )
RC_NEED_br0="net.eth0"
brctl_br0=( "setfd 0" "sethella 1" "stp off" )
routes_br0=( "default gw 192.168.0.254" )
or using dhcp:
```

```
bridge_br0="eth0"
config_eth0=( "null" )
config_br0=( "dhcp" )
dhcpcd_br0="-t 10"
RC_NEED_br0="net.eth0"
brctl_br0=( "setfd 0" "sethella 1" "stp off" )
```

Use net.eth0 config as base for net.br0:

```
sudo cp /etc/init.d/net.eth0 /etc/init.d/net.br0
```

Restart the network:

```
/etc/init.d/net.br0 restart
```

.. Configure a network bridge in Ubuntu

First, install the necessary packages for the bridge management:

```
emerge net-misc/bridge-utils sys-apps/usermode-utilities
```

Create the bridge:

```
sudo brctl addbr br0
```

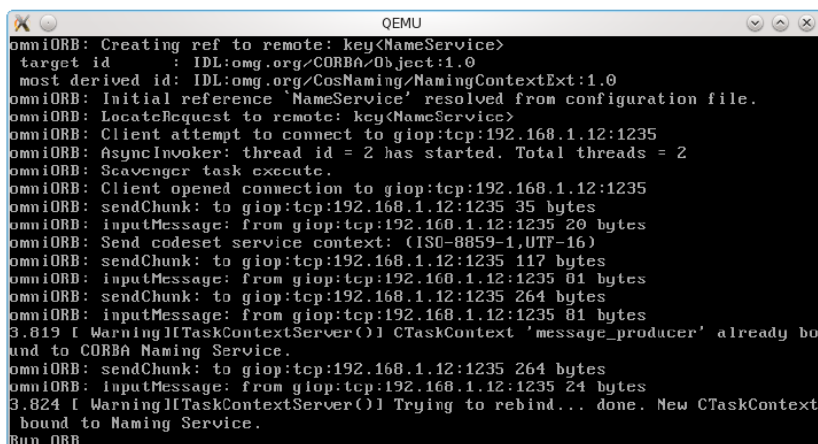
. Running TCP/IP applications on QEMU

At this point, you should be able to see, using ifconfig, a new interface (called br0) with the previously specified IP. You can now start QEMU by typing (as root) the next command:

```
qemu -netnic,macaddr=00:80:7F:22:61:77,model=ne2k_isa -nettap,script=$BUILD_PREFIX/scripts/files/qemu-ifup -kernel application.exe
$BUILD_PREFIX/scripts/files/qemu-ifup contains:
```

```
#!/bin/sh
TUN_DEV=$1
TUN_HOST=10.0.2.2
/sbin/ifconfig $TUN_DEV $TUN_HOST
exit 0
```

Figura 2: Message producer tutorial





. RTEMs network examples

You can check your network configuration on QEMU using RTEMs network examples: <http://www.rtems.info/ftp/pub/rtems/4.10.1/network-demos-4.10.1.tar.bz2>.

Figura 3: RTEMs netdemo on QEMU

```

... failed
Probing 'ep0'3C509: attach() called.
3C509: isa_probe() looking for a card...
3C509: isa_probe() fail to find a board.
... failed
Probing 'ne1' network device 'ne1' <00:80:7F:22:61:77> initialized on port 0x300,
irq 9
... seemed to work
Network initialized
Destination      Gateway/Mask/Hw  Flags    Refs    Use  Expire  Interface
default         10.0.2.2         UGS      0       0    0      ne1
10.0.0.0         255.0.0.0       U        0       0    1      ne1
10.0.2.2         10.0.2.2        UHL      1       0    1      ne1
127.0.0.1        127.0.0.1       UH       0       0    0      lo0
Initiating test
Create socket.
Bind socket.
Listen.
Accept.
Create socket.
Bind socket.
Listen.
Accept.

```

Using TSIM for RTEMs LEON2 (SPARC v8) BSP

TSIM is an instruction-level simulator capable of emulating LEON-based computer systems developed by Aeroflex Gaisler AB . TSIM can be run in stand-alone mode, or connected through a network socket to the GNU gdb debugger.

You can run a Rock application with TSIM in stand-alone mode by typing:

```
tsim -leon application.exe
tsim > go
```

For large binaries, different amount of simulated RAM(kbyte) can be set with:

```
tsim -leon -ram 32768 application.exe ( for 32768 K RAM )
tsim > go
```

. GDB usage

Connected to gdb, TSIM acts as a remote target and supports all gdb debug requests. The communication between gdb and TSIM is performed using the gdb extended-remote protocol.

```
tsim -leon application.exe -gdb
```

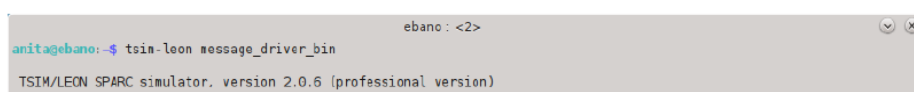
Then launch gdb on the "application.exe" executable, and connect to TSIM:

```
gdb application . exe
( gdb ) target remote localhost :1234
```

Then you can use gdb normally. For example, type "run" to launch the kernel:

```
( gdb ) run
```

Figura 4: Message driver tutorial on TSIM-LEON2



```
Copyright (C) 2001, Gaisler Research - all rights reserved.  
For latest updates, go to http://www.gaisler.com/  
Comments or bug-reports to tsim@gaisler.com
```

```
serial port A on stdin/stdout  
allocated 4096 K RAM memory, in 1 bank(s)  
allocated 2048 K ROM memory  
icache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)  
dcache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)  
section: .text, addr: 0x40000000, size 584832 bytes  
section: .data, addr: 0x4008ec80, size 15872 bytes  
section: .jcr, addr: 0x40092a80, size 4 bytes  
read 1153 symbols  
tsim> go  
resuming at 0x40000000  
[567993600010000] Message from MessageDriver  
[567993600010000] MESSAGE FROM MESSAGEDRIVER  
█
```

Notas al pie

... RTEMS

If you are more interested on how RTEMS works, please read Getting Started with RTEMS.

... packages

Specific versions can be modified in rock/build/scripts/files/rtems_ver

... AB

<http://www.gaisler.com/>

1. http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/ruby_start.ogv
2. http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/message_consumer.ogv
3. http://roble.datsi.fi.upm.es/~anita/rock/rockOnRTEMS/media/message_manager.ogv