Freedom Embedded

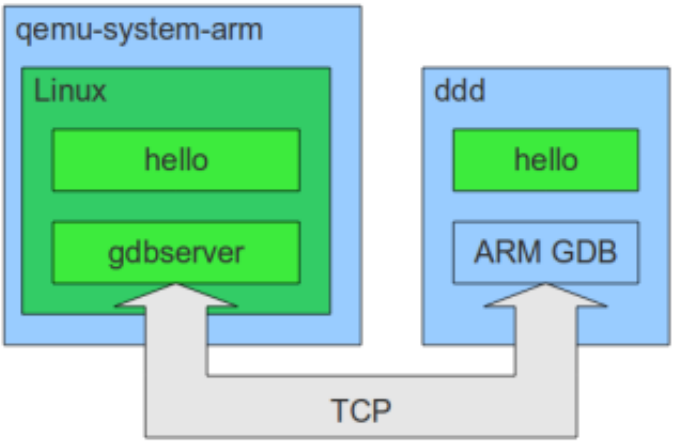
Balau's technical blog on open hardware, free software and security

Debugging ARM programs inside QEMU

Posted on 2010/08/17

50

Recently I wanted to debug a Linux program running inside an ARM system emulated with QEMU (http://wiki.qemu.org/). I went into some troubles, so I'm going to write here the procedure that worked for me. I wanted to use gdbserver to run a program inside QEMU, and then connect to it from a GDB instance running on my PC, using a TCP link. gdbserver is a piece of software that implements some of the GDB (http://www.gnu.org/software/gdb/) functionalities (the debugging stubs) and then offers the possibility to connect to a full GDB instance through the network (or through a serial port). What I wanted to obtain is illustrated in the following figure.



(https://balau82.files.wordpress.com/2010/08/qemu-gdbserver.png)
Using gdbserver inside QEMU

The color blue indicates software compiled to run on my Ubuntu (http://www.ubuntu.com/) PC (32-bit x86) while the color green indicates software compiled to run on ARM. qemu-system-arm is the software that emulates a VersatilePB (http://infocenter.arm.com/help/topic/com.arm.doc.dui0224i/index.html) platform; I tried to use the one that can be installed through Ubuntu repositories (the package is qemu-kvm-extras) but it froze running the last version of Linux (2.6.35). For this reason I decided to compile the upstream version and use that one. The GDB server and "client" come from the CodeSourcery compiler collection for ARM (http://www.codesourcery.com/sgpp/lite/arm/portal/release1293), as well as

the compiler used to cross-compile the software for ARM. The program I'll debug in this example is the simple GNU Hello (http://www.gnu.org/software/hello/), that doesn't do very much beyond printing "Hello World", but is a nice example of cross-compilation with GNU Autotools.

Prerequisites

In order to follow this procedure, I installed:

- CodeSourcery GNU/Linux toolchain for ARM
- The native x86 toolchain (build-essentials package in Ubuntu) to compile QEMU
- DDD as a graphic interface to the debugger
- cpio, an utility to create the Linux filesystem image.
- the package libncurses5-dev to run the menu configurations of both Linux kernel and Busybox
- libsdl-dev and zlib1g-dev to compile QEMU

```
1 $ sudo apt-get install build-essential ddd cpio libncurses5-dev libsdl-dev
2 $ wget http://www.codesourcery.com/sgpp/lite/arm/portal/package6490/publi
3 $ chmod +x arm-2010q1-202-arm-none-linux-gnueabi.bin
4 $ ./arm-2010q1-202-arm-none-linux-gnueabi.bin
```

I installed the toolchain in the default directory "~/CodeSourcery". The gdbserver executable in my case can be found at the path "/home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/usr/bin/gdbserver".

Note that the following procedure is run in a dedicated folder, and no root access is required from now on. At the end of the procedure about 1 Gigabyte of hard disk space was used.

Linux Kernel

First of all, I took the new kernel version from the official repositories (http://www.kernel.org/).

```
1 $ wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.35.tar.bz2 (
2 $ tar xjf linux-2.6.35.tar.bz2
3 $ cd linux-2.6.35/
4 $ make ARCH=arm versatile_defconfig
5 $ make ARCH=arm menuconfig
```

When the menu appears, I go into the "Kernel Features" section and enable EABI support; then I exit (saving the configuration) and compile:

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- all
$ cd ..
```

The result is a compressed kernel image in "./linux-2.6.35/arch/arm/boot/zImage".

Busybox

Next, I take the latest version of Busybox (http://www.busybox.net/); in a previous tutorial (https://balau82.wordpress.com/2010/03/27/busybox-for-arm-on-qemu/) I compiled it statically, but this time I will not, because gdbserver (that I plan to use) needs shared libraries anyway.

```
$ wget http://busybox.net/downloads/busybox-1.17.1.tar.bz2 (http://busybo
$ tar xjf busybox-1.17.1.tar.bz2
$ cd busybox-1.17.1
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- defconfig
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- install
$ cd ..
```

The result is the folder "busybox-1.17.1/_install", containing a minimal root filesystem that lacks shared libraries.

QEMU

I recompiled from source only the QEMU binaries needed to emulate an ARM system.

```
$ wget http://download.savannah.gnu.org/releases/qemu/qemu-0.12.5.tar.gz
$ tar xzf qemu-0.12.5.tar.gz
$ cd qemu-0.12.5
$ ./configure --enable-sdl --disable-kvm --enable-debug --target-list=arm
$ ./make
$ cd ..
```

The relevant result is the program "./qemu-0.12.5/arm-softmmu/qemu-system-arm" that will be used to emulate the VersatilePB platform.

GNU Hello

This package needs to be configured for cross-compilation; turns out it's very easy to do: it needs just the prefix of the cross-compiler.

```
$ wget http://ftp.gnu.org/gnu/hello/hello-2.6.tar.gz
$ tar xzf hello-2.6.tar.gz
$ cd hello-2.6
$ ./configure --host=arm-none-linux-gnueabi
$ make
$ cd ..
```

The result is the ARM-executable "hello-2.6/src/hello".

Complete the filesystem

All the ARM binaries involved (busybox, gdbserver, hello) need shared libraries. The Codesourcery toolchain offers these libraries in a subfolder of its installation. In my case it's "/home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/". In order to discover what are the needed libraries I used the readelf tool distributed in the CodeSourcery toolchain. In particular, I ran:

```
$ arm-none-linux-gnueabi-readelf hello-2.6/src/hello -a |grep lib
 [Requesting program interpreter: /lib/ld-linux.so.3]
 0x00000001 (NEEDED)                     Shared library: [libgcc_s.so.1]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
00010694  00000216 R_ARM_JUMP_SLOT   0000835c   __libc_start_main
 2: 0000835c     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2
 89: 0000844c     4 FUNC    GLOBAL DEFAULT   12 __libc_csu_fini
 91: 0000835c     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
 101: 00008450   204 FUNC    GLOBAL DEFAULT   12 __libc_csu_init
 000000: Version: 1  File: libgcc_s.so.1  Cnt: 1
 0x0020: Version: 1  File: libc.so.6  Cnt: 1
```

The hello binary requires three shared libraries: "ld-linux.so.3", "libgcc_s.so.1" and "libc.so.6". I executed this command for all three binaries, and I copied the required libraries into the Busybox filesystem, together with the gdbserver executable and the hello executable.

```
$ cd busybox-1.17.1/_install
$ mkdir -p lib
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueal
$ cp ../../hello-2.6/src/hello usr/bin/
```

```
10  $ cd ../../
```

For my experiment I need a working network from the guest ARM system side, so I prepared an initialization script to enable it. Extending from my previous tutorial (https://balau82.wordpress.com/2010/03/27/busybox-for-arm-on-qemu/), here is the script "rcS" that i used.

```
1  #!/bin/sh
2  mount -t proc none /proc
3  mount -t sysfs none /sys
4  /sbin/mdev -s
5  ifconfig lo up
6  ifconfig eth0 10.0.2.15 netmask 255.255.255.0
7  route add default gw 10.0.2.1
```

Next, I copy rcS it inside the "etc/init.d" directory of the Busybox filesystem and create a compressed filesystem image:

```
1  $ cd busybox-1.17.1/_install
2  $ mkdir -p proc sys dev etc etc/init.d
3  $ cp ../../rcS etc/init.d
4  $ chmod +x etc/init.d/rcS
5  $ find . | cpio -o --format=newc | gzip > ../../rootfs.img.gz
6  $ cd ../../
```

Running and debugging

Now I have put everything in place:

- A compressed kernel image
- QEMU
- A compressed filesystem image containing:
  - Busybox
  - rcS initialization script
  - GNU Hello binary compiled for ARM
  - gdbserver for ARM
  - Required shared libraries

To run the plaform the command line is:

```
1  $ ./qemu-0.12.5/arm-softmmu/qemu-system-arm -M versatilepb -m 128M -kerne
```

The “redir” option will redirect any TCP communication from port 1234 of my Ubuntu PC to port 1234 of the guest ARM system. The system will boot, and a console can be opened with root access. Inside the bash prompt, I run the debugging server:

```
# gdbserver --multi 10.0.2.15:1234
```

This command will launch a server that waits for a GDB connection from port 1234. On my PC I open the debugger:

```
$ ddd --debugger arm-none-linux-gnueabi-gdb
```

It is also possible to run the arm-none-linux-gnueabi-gdb command alone or connected to another front-end. In order to debug the remote program, I need to tell GDB to consider the ARM shared libraries instead of the local ones (that are for 32-bit x86); otherwise on execution the debugger will complain that the libraries don’t match.

```
set solib-absolute-prefix nonexistantpath
set solib-search-path /home/francesco/CodeSourcery/Sourcery_G++_Lite/arm-
file ./hello-2.6/src/hello
target extended-remote localhost:1234
set remote exec-file /usr/bin/hello
break main
run
```

At this point the debugging goes on as usual.

Tagged: *ARM*, *codesourcery*, *debug*, *gdb*, *gnu*, *helloworld*, *howto*, *kernel*, *linux*, *qemu*, *tutorial*
Posted in: *Embedded (https://balau82.wordpress.com/category/software/embedded-software/)*

50 Responses “Debugging ARM programs inside QEMU” →

M.Vivek

2011/02/05

Hi Balau,

I ran the steps that you have outlined . But, on doing “run” in the gdb console of ddd, the QEMU
session exits with :
****
…
Please press Enter to activate this console.
/ # gdbserver –multi 10.0.2.15:1234
Listening on port 1234

QEMU: Terminated via GDBstub
***
What could be the issue ?
Regards,
M.Vivek

Balau

2011/02/05

I get your same error if I run QEMU with debug option enabled (for example “-s”). In this way, when you connect with gdb from the outside, you are actually connected to QEMU and not to the gdbserver running inside it. If you did not enable QEMU debug options, maybe you have a QEMU option enabled by default, try using a different port (1235) for both gdbserver and the ddd “target extended-remote” command.

Ram

2011/03/12

Hi,
I followed your step, but last step qemu just hangs. System doesn’t boot.

Balau

2011/03/13

Do you mean that when you run the “qemu-system-arm” command it doesn’t show anything about the Linux kernel booting? Or do you see some messages but at the end there are some errors and it doesn’t show the command shell?

Ram

2011/03/13

After qemu-system-arm command qemu screen appears, but it doesn't show anything.(Just black screen), no message about booting at all.
I tried booting linux kernel "linux-0.2.ing" provided by qemu site with rootfile system provided by the qenu itself, it works fine.

Ram

2011/03/13

I was trying that with kernel 2.6.37, now I tried with 2.6.35. Now I get error

Kernel Panic – Not syncing : Attempted to kill init !

Balau

2011/03/13

I think you mean the "arm-test-0.2.tar.gz" test containing the arm_root.img file.
Anyway, when you compile the kernel like in my test, what does it say when you run the command "file ./linux-2.6.35/vmlinux"?
Are you using the same version of Linux kernel, QEMU and CodeSourcery toolchain that I'm using?

Balau

2011/03/13

This usually happens when I forget to enable EABI support in the Kernel configuration, so that the kernel tries to execute Busybox binaries and fails, exiting the "init" program.

Ram

2011/03/13

Hey I got it. I didn't enabled EABI support in Kernel Features.

Thank you.

Best Regards
Ram

Ram

2011/05/24

Hi,
Is it possible to run dynamically linked binary in qemu?
I am trying to run DirectFB test applications in qemu and I have copid all required libraries (cross-compiled for arm) in /usr/lib, but still I am getting error.

/bin/sh : ./test :not found

Balau

2011/05/24

Have you populated also the C standard libraries and the Linux dynamic loader (ld-linux.so*)? They are usually provided by the toolchain. For example in CodeSourcery these libraries are in "`Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib`" and in "`Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/usr/lib`"
You can see the libraries that a program need by running "readelf -d -l" (or the cross-compiler version of it) on the executable and noting the "program interpreter" and "shared library" lines.
In your case, maybe the "sh" interpreter or the "test" program needs some libraries that have not been copied.

Ram

2011/05/25

Hi,
I have copied all required libraries in /usr/lib (using the rootfs built according to your post for nfs set up), but still getting same error.

I can't execute even simple "hello world" program in qemu-system-arm, unless binary is compiled with -static option.

Is there anything that need to be done at time of building rootfs or kernel image, to enable execution of dynamically linked binaries?

Ram

2011/05/25

Hi,

I got the problem
(ld-linux.so*) need to be in /lib, along with its real library it is pointing to.

Earlier I copied it in /usr/lib

Regards,

Mandar Vaidya

2011/09/21

/bin/sh: arm-none-linux-gnueabi-gcc: command not found

error get while running the make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- all command
please send reply

Balau

2011/09/21

You probably don't have the toolchain in your PATH.
When you installed CodeSourcery, the installation should have added in your login scripts some lines that add the toolchain directory to the PATH environmental variable, so that the next time you open a terminal they are set correctly and you can run it.

By default the binaries are installed in "`~/CodeSourcery/Sourcery_G++_Lite/bin`", so you should check if that directory is contined in the PATH, by running "`echo $PATH`" in the terminal. To add it manually, the command is:

```
export PATH="~/CodeSourcery/Sourcery_G++_Lite/bin:${PATH}"
```

If you installed CodeSourcery into another directory then you need to use that directory instead.

ericwain

2011/09/27

Hi Balau,
After Linux booting, I follow your step to do debug. But I get this error message when I execute
target extended-remote localhost:1234.
warning: unrecognized item "timeout" in "qSupported" response
————————————————————
My ubuntu IP is 192.168.104.3, so I change IP of rcS file to 192.168.104.4

#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s
ifconfig lo up
ifconfig eth0 192.168.104.4 netmask 255.255.255.0
route add default gw 192.168.104.1
(After booting, it can not ping to 192.168.104.3, why???)

In Qemu:
qemu-system-arm -M versatilepb -m 128M -kernel linux-2.6.35.9/arch/arm/boot/zImage -initrd rootfs.img.gz -append "root=/dev/arm rdinit=/sbin/init" -redir tcp:1234::1234

gdbserver –multi 192.168.104.4:1234

In GDB:
(gdb) set solib-absolute-prefix nonexistantpath
(gdb) set solib-search-path ~/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/
(gdb) file ./hello-2.6/src/hello
Reading symbols from /home/erictsai/workspace/qemu/hello-2.6/src/hello…done.
(gdb) target extended-remote localhost:1234
Remote debugging using localhost:1234
Ignoring packet error, continuing…
warning: unrecognized item "timeout" in "qSupported" response
Ignoring packet error, continuing…
Ignoring packet error, continuing…
Ignoring packet error, continuing…
Ignoring packet error, continuing…

any suggestions for this issue?

Thank you.

Balau

2011/09/27

Your Ubuntu IP does not mean you have to change your rcS. Your guest system is still seeing a "10.0.*.*" network created by QEMU; check this page for more info: QEMU Networking

ericwain

2011/09/28

Hi balau,
According to your reply, I have solved this problem. Thank you. It can trace hello problem now. But when I trace setlocale (LC_ALL, "") in hello program, gdb will show:

Cannot access memory at address 0x0

Program received signal SIGSEGV, Segmentation fault.
0xe7f001f0 in ?? ()

How can it trace these system calls like setlocale or fprintf?

Again, Can it debug Linux kernel (like start_kernel) under this environment?

Balau

2011/10/02

The SIGSEGV could be a problem in shared libraries.
Try to compile the "hello" program with "`CFLAGS=-static make`" and see if it gives you the same error.
If it runs OK, then check if you are copying the correct libraries into the busybox filesystem.

To debug the kernel, run `qemu-system-arm` with "`-s -S`" options but without the "`-redir tcp:1234::1234`" option, or at least use a different port, because "`-s -S`" options start QEMU with a GDB server listening to port 1234.
When QEMU is opened it is freezed waiting for a connection, then you can run "`ddd --debugger arm-none-linux-gnueabi-gdb`", and inside the gdb prompt you can run something like:
`file linux-2.6.35/vmlinux` (to get the Linux debugging symbols)
`target remote localhost:1234` (to connect to QEMU)
`break start_kernel`
`continue`

ericwain

2011/10/03

Hi Balau,
I set the wrong shared library path.. >_<..
Hello program can be debugged correctly now .
Thank you.

john

2011/11/13

Hi Balau,

Nice tutorial you have there. During the last part when i use run in gdb i bump into some warnings:

(gdb) run
warning: `/lib/libgcc_s.so.1′: Shared library architecture unknown is not compatible with target architecture arm.
warning: .dynamic section for “/lib/libgcc_s.so.1″ is not at the expected address (wrong library or version mismatch?)
warning: `/lib/libc.so.6′: Shared library architecture unknown is not compatible with target architecture arm.
warning: .dynamic section for “/lib/libc.so.6″ is not at the expected address (wrong library or version mismatch?)

is these warning crictical? or i missed something in between =)

Balau

2011/11/13

It seems that gdb is trying to use the libraries of your host PC (which are usually for x86 architectures) instead of the CodeSourcery ones (which are for ARM).

I think that if you use it anyway, the debugger could crash when you reach a library function such as printf.

The “solib-absolute-prefix” command with a non-existant path as the argument should be used to avoid the host libraries, then the “solib-search-path” command should point to the right libraries. Are you sure you ran these commands correctly?

john

2011/11/14

Not so sure though.

GNU DDD 3.3.12 (i386-redhat-linux-gnu), by Dorothea Lütkehaus and Andreas Zeller.
Copyright © 1995-1999 Technische Universität Braunschweig, Germany.
Copyright © 1999-2001 Universität Passau, Germany.
Copyright © 2001 Universität des Saarlandes, Germany.
Copyright © 2001-2004 Free Software Foundation, Inc.
(gdb) pwd
Working directory /home/student.
(gdb) cd verilog_examples/A

verilog_examples/ARM
verilog_examples/AT510
(gdb) cd verilog_examples/ARM/QEMU/
(gdb) pwd
Working directory /home/student/verilog_examples/ARM/QEMU.
(gdb) cd debugging-arm-programs-inside-qemu
(gdb) set solib-absolute-prefix nonexistant
(gdb) set solib-search-path /home/student/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/
(gdb) file ./hello-2.6/src/hello
(gdb) target extended-remote localhost:1234
(gdb) set remote exec-file /usr/bin/hello
(gdb) break main
Breakpoint 1 at 0x8b3c: file hello.c, line 52.
(gdb) run
warning: `/lib/libgcc_s.so.1′: Shared library architecture unknown is not compatible with target architecture arm.
warning: .dynamic section for "/lib/libgcc_s.so.1″ is not at the expected address (wrong library or version mismatch?)
warning: `/lib/libc.so.6′: Shared library architecture unknown is not compatible with target architecture arm.
warning: .dynamic section for "/lib/libc.so.6″ is not at the expected address (wrong library or version mismatch?)

Breakpoint 1, main (argc=1, argv=0xbeefeeb4) at hello.c:52
(gdb)

Balau

2011/11/14

The commands seem OK if "/home/student/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/" contains "libgcc_s.so.1″ and the other libraries.
Are you sure you are using the ARM toolchain GDB and not your PC's classic gdb?
Try to run "arm-none-linux-gnueabi-gdb" without DDD and see if the error is the same.
The problem is that if you use ARM binutils with x86 binaries or x86 binutils with ARM binaries, the architecture is seen as "unknown".
Try also to run "arm-none-linux-gnueabi-objdump -f …" on the library files that you copied in "_install/lib". If the architecture is "UNKNOWN!" then you copied the wrong libraries.

john

2011/11/14

Hi,

Just tested the objdump. the library files seems to be correct

[student@fedora lib]$ arm-none-linux-gnueabi-objdump -f ld-linux.so.3

ld-linux.so.3: file format elf32-littlearm
architecture: arm, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x000007a0

[student@fedora lib]$ arm-none-linux-gnueabi-objdump -f lib
libc.so.6* libdl.so.2* libgcc_s.so.1* libm.so.6*
[student@fedora lib]$ arm-none-linux-gnueabi-objdump -f libc.so.6

libc.so.6: file format elf32-littlearm
architecture: arm, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0001595c

[student@fedora lib]$ arm-none-linux-gnueabi-objdump -f libgcc_s.so.1

libgcc_s.so.1: file format elf32-littlearm
architecture: arm, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x000028ec

john

2011/11/14

Hi,

When i used the command without ddd, i get the same error too. @.@

[student@fedora ~]$ arm-none-linux-gnueabi-gdb
GNU gdb (Sourcery G++ Lite 2011.03-41) 7.2.50.20100908-cvs
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type “show copying”
and “show warranty” for details.
This GDB was configured as “–host=i686-pc-linux-gnu –target=arm-none-linux-gnueabi”.
For bug reporting instructions, please see:
.
(gdb) pwd
Working directory /home/student.
(gdb) cd v

```
vcs_lab/ verilog_examples/
(gdb) cd verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/
Working directory /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu.
(gdb) set solib-absolute-prefix nonexistantpath
(gdb) set solib-search-path /home/student/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/
(gdb) file ./hello-2.6/src/hello
Reading symbols from /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello…done.
(gdb) target extended-remote localhost:1234
set extended-remote localhoslocalhost:1234: Connection timed out.
(gdb) set extended-remote localhost:1234
No symbol "extended" in current context.
(gdb) target extended-remote localhost:1234
localhost:1234: Connection timed out.
(gdb) set remote exec-file /usr/bin/hello
(gdb) break main
Breakpoint 1 at 0x8b3c: file hello.c, line 52.
(gdb) run
Starting program: /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello
Don't know how to run. Try "help target".
(gdb) run
Starting program: /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello
Don't know how to run. Try "help target".
(gdb) file ./hello-2.6/src/hello
Load new symbol table from "/home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello"? (y or n) y
Reading symbols from /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello…done.
(gdb) target extended-remote localhost:1234Remote debugging using localhost:1234
(gdb) set remote exec-file /usr/bin/hello
(gdb) break main
Note: breakpoint 1 also set at pc 0x8b3c.
Breakpoint 2 at 0x8b3c: file hello.c, line 52.
(gdb) run
Starting program: /home/student/verilog_examples/ARM/QEMU/debugging-arm-programs-inside-qemu/hello-2.6/src/hello
warning: `/lib/libgcc_s.so.1': Shared library architecture unknown is not compatible with target architecture arm.
warning: .dynamic section for "/lib/libgcc_s.so.1" is not at the expected address (wrong library or version mismatch?)
warning: `/lib/libc.so.6': Shared library architecture unknown is not compatible with target
```

architecture arm.
warning: .dynamic section for "/lib/libc.so.6" is not at the expected address (wrong library or version mismatch?)

Breakpoint 1, main (argc=1, argv=0xbebb1eb4) at hello.c:52
52 program_name = argv[0];
(gdb)

Balau

2011/11/14

I'm sorry but I think I can't solve this problem.
It may be something different between your RedHat/Fedora distribution and my Debian/Ubuntu, but I don't know what it is.

john

2011/11/14

Ohh ok. Thanks for the help =) will let you know if i managed it

Mandar Vaidya

2011/11/16

sir,
how to load(install) the free rtos on qemu for cortex-m3 arm processor.

Balau

2011/11/16

I would like to do it, too, but I never found the time.
In my opinion I would start with this setup:

Using CodeSourcery bare metal toolchain for Cortex-M3

Hoping that freeRTOS has good support for that Luminary Micro core, I would compile one of the demo programs and it will create a binary image that can be passed on with the "-kernel" or with the "-pflash" option of QEMU.
As I said, I never did it so it is just an idea and I don't know if there are some details that make it impossible/hard to do.

zhenningjohn

2011/12/01

Managed to compile and run it. I just need the latest version of busybox and qemu ^_^

Satish Kumar

2013/12/27

Hi ,
I am trying the same as above article, I got following error after giving command gdbserver –multi 10.0.2.15:1234
“gdbserver : error while loading shared libraries: libthread_db.so.1:cannot open shared object file : no such file or directory”

do i have to change any parameter inmy system to connect to Remote Pc to Local Pc/Ubuntu system.
what is exact use of Ip addres 10.0.2.15 [Gone through Qemu networking article but I got less]
If I want to debug kernel how I can do/changes I have to do?
Can you please guide for the above.
Thank you in advance

Regards
gsatish.net

Balau

2013/12/27

About your first problem:

In my post I said that `hello` program needs some shared libraries, but it seems also `gdbserver` needs some. Maybe we use different Sourcery versions and your version needs another library. You can check with something like:

```
$ arm-none-linux-gnueabi-readelf "busybox-1.17.1/_install/usr/bin/gdbserver" -a |grep lib
```

When you are still preparing the `_install` directory, you probably need to do something like this to copy another shared library:

```
$ cp .../Sourcery_G++_Lite/arm-none-linux-gnueabi/libc/lib/libthread_db* lib/
```

About your networking doubts:

QEMU creates a virtual network that is visible only by guest systems.
If you have two real PCs on a real ethernet, you need to use the IP address of the system where gdbserver is run both in the gdbserver command and in GDB client.
So if your remote PC to debug has IP address `192.168.0.101`, you run:

```
$ gdbserver –multi 192.168.0.101:1234
```

and on your system, inside GDB, you run:

```
(gdb) target extended-remote 192.168.0.101:1234
```

About kernel debugging, I don't have much experience but in QEMU you can use "-s -S" options and then connect with arm-none-linux-gnueabi-gdb using "`target remote localhost:1234`" like I do in .

Satish Kumar

2013/12/28

Hi
Thank you for your quick reply for my post
I got solution for my first problem ,adding gdb server libraries in compressed root file system..

I am trying to debug kernel using commands given in one of your comments
when I am giving "file /vmlinux" on gdb . it is showing as "no debugging symbols found." I tried might kernel need to copy any shared libraries form arm-none-linux-gnueabi-readelf , but it is not showing any ".so". Can you please guide what I am missing here. I am using qemu-arm-system with -s -S options and qemu is freezed there as u mention.
I am using port 1235 instaed of 1234.

Thank you in advance.
–Satish.G

Balau

2013/12/28

Kernel doesn't need shared libraries.
When you configure the kernel, you run "make menuconfig" (needs ncurses devel package to be installed) and then go into "Kernel hacking" section and enable "Compile the kernel with debug info" (CONFIG_DEBUG_INFO option).
I don't know what you mean by "I am using port 1235 instead of 1234" because it contrasts with "-s -S" options. If you run qemu with "–help" it shows:

-gdb dev wait for gdb connection on 'dev'
-s shorthand for -gdb tcp::1234

So to use 1235 you need to run qemu with "-S -gdb tcp::1235" option, and then you can do "target remote localhost:1235" in gdb.

Satish Kumar

2013/12/30

Hi balau,
I can debug kernel now from qemu emulated system.
Thank you for continuous support

Satish.G

Satish Kumar

2013/12/30

When I press "con" [continue] button in debugger getting error as "Failed to execuate /sbin/init , Kernel panic -not syncing no init found." . It is looking for kernel files in home path and fila to get those files. I am providing following commands :
"qemu-system-arm -s -S -M versatilepb -m 128M -kernel -initrd -append "root=/dev/ram rdinit=/sbin/init" . with this command qemu opening and showing stopped on right corner of qemu. on other terminal i am opening debugger with command "ddd –debugger arm-none-linux-gnueabi-gdb" it is opening debugger window . providing "file , target remote localhost:1234 , break start_kernel and continue. so when I am pressing con button it is showing function names but qemu still showing as "stopped" on right corner and when I press run getting error as "failed to execuate /sbin/init: kernel panic -not syncing no init found". so can you please guide where is mistake i am doing. do i providing wrong commands or have to add port no. to qemu-system-arm?
Thank you in advance.

Satish Kumar

2013/12/30

Hi Balau,
i got solution for above mentioned problem, that is because of rootfs error [based on linux doc/init.txt]. now qemu is entering into # prompt and I am providing "gdbserver –multi 10.0.2.15:1234" but on gdb when we give con button it is showing as " program received signal SIGINT, interrupt" .

I am giving options as “qemu-system-arm -S -gdb tcp::1235″
Can u please locate error .
Thank you .

Satish Kumar

2014/01/05

Hi balau,
I got the output to debug kernel inside qemu using following commands:
“qemu-system-arm -M versatilepb -m 128M -kernel zImage -initrd busybox-1.16.0/rootfs.img.gz -append “root=/dev/ram rdinit=/sbin/init” -s -S ”
zImage is compiled for versatilepb board with EABI and kernel debug info enable in menuconfig. initrd is created based on your post/article with busybox with qemu.
on gdb side: arm-none-linux-gnueabi-gdb /vmlinux
target remote:1234
shows message as “remote debugging to 1234″

set breakpoints: break start_kernel
break pidmap_init
continue/next

so Now I can debug kernel running inside qemu.

Thank you
Satish.G
gsatish.net

Balau

2014/01/06

Glad you solved your problems. Sorry for not being able to respond, but I was on holiday.

Ijaz

2014/03/17

Hi balau

Thanks for these very informative articles here. They really help budding embedded software folks get going.

To begin with, Im working with the PowerPC architecture here and my host PC is debian wheezy. Ive installed emdebian toolchain for PowerPCon it. The linux kernel has been cross compiled to enable KGDB. In addition Ive compiled gdbserver from source and placed it to go with my final busybox image. Along with it, I also placed the libraries on which gdbserver depends.

```
powerpc-linux-gnu-readelf ./gdbserver -a | grep lib
[Requesting program interpreter: /lib/ld.so.1]
0x00000001 (NEEDED) Shared library: [libdl.so.2]
0x00000001 (NEEDED) Shared library: [libc.so.6]
1005843c 00002615 R_PPC_JMP_SLOT 00000000 __libc_start_main + 0
38: 00000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_2.0 (2)
100: 1000908c 108 FUNC LOCAL DEFAULT 13 handle_qxfer_libraries_sv
128: 1000d198 908 FUNC LOCAL DEFAULT 13 handle_qxfer_libraries
498: 1002a08c 2120 FUNC LOCAL DEFAULT 13 linux_qxfer_libraries_svr
641: 100603cc 4 OBJECT LOCAL DEFAULT 28 libthread_db_search_path
690: 100346f0 4 FUNC GLOBAL DEFAULT 13 __libc_csu_fini
800: 00000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_
961: 10034700 180 FUNC GLOBAL DEFAULT 13 __libc_csu_init
000000: Version: 1 File: libdl.so.2 Cnt: 2
0x0030: Version: 1 File: libc.so.6 Cnt: 8
```

Ive also added the snipped that you showed to my rcS script.

Ive been able to boot QEMU emulating an MPC8544DS system, with this busybox as my rootfs, using this as at the command prompt:

```
../Qemu1.7.0_bins/bin/qemu-system-ppc -M mpc8544ds -m 512 -kernel zImage -
nographic -initrd busyboxfs_wGDB.img -append "root=/dev/ram
rdinit=/sbin/init" -redir tcp:1234::1234
```

However from within the target, I am unable to run gdbserver:

```
/ # gdbserver --multi 10.0.2.15:1234
-/bin/sh: gdbserver: not found
```

Im sure the gdbserver has been properly cross compiled. Im not sure if gdbserver also needs to be statically compiled? I compiled gdbserver using the classic autotools to configure it to use the cross compiler using the 'host' option within configure. Im running out of ideas what could I have done wrong.

Keen to hear.

Balau

2014/03/17

From within the target, try to give the following commands:
To check if it's in the PATH:
```
# which gdbserver
```
From there on, I assume you have copied it in `/usr/bin/`.
To check if the execution bit is set:
```
# ls -l /usr/bin/gdbserver
```
To check the architecture:
```
# file /usr/bin/gdbserver
```
To check that dynamic libraries are found:
```
# ldd /usr/bin/gdbserver
```
To try to run it and see some error messages you can use /lib/ld.so.1 (the name might be different) as a program:
```
# /lib/ld*.so.1 /usr/bin/gdbserver
# /lib/ld*.so.1 --list /usr/bin/gdbserver
# /lib/ld*.so.1 --verify /usr/bin/gdbserver
```

Ijaz

2014/03/22

Hi balau
Thanks for the prompt reply. Would like to say that by default, when I build busybox, theres no '`lib`' directory within '`_install`'. I presume we have to create one ourselves, just like we create `proc, sys, etc` etc. Correct?
After that I copied the dependent libraries from the relevant toolchain installation directory into it. Additionally I copied `gdbserver` (compiled for PowerPC earlier) into `/usr/bin` and created the busybox image. Heres the boot log for QEMU:

```
Using MPC8544 DS machine description
Memory CAM mapping: 256/256/0 Mb, residual: 0Mb
Linux version 2.6.32.32wKGDB (aijazbaig1@debianbox) (gcc version 4.4.5
(Debian 4.4.5-8) ) #4 SMP Sun Feb 23 13:00:26 IST 2014
Found initrd at 0xc2000000:0xc2464400
CPU maps initialized for 1 thread per core
bootconsole [udbg0] enabled
setup_arch: bootmem
mpc85xx_ds_setup_arch()
Found FSL PCI host bridge at 0x00000000e0008000. Firmware bus number: 0->255
PCI host bridge /pci@e0008000 ranges:
MEM 0x00000000c0000000..0x00000000dfffffff -> 0x00000000c0000000
IO 0x00000000e1000000..0x00000000e100ffff -> 0x0000000000000000
/pci@e0008000: PCICSRBAR @ 0xfff00000
MPC85xx DS board from Freescale Semiconductor
arch: exit
Zone PFN ranges:
```

```
DMA 0x00000000 -> 0x00020000
Normal 0x00020000 -> 0x00020000
HighMem 0x00020000 -> 0x00020000
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
0: 0x00000000 -> 0x00020000
MMU: Allocated 1088 bytes of context maps for 255 contexts
PERCPU: Embedded 8 pages/cpu @c0a50000 s8328 r8192 d16248 u65536
pcpu-alloc: s8328 r8192 d16248 u65536 alloc=16*4096
pcpu-alloc: [0] 0
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 130048
Kernel command line: root=/dev/ram rdinit=/sbin/init
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 508416k/524288k available (5948k kernel code, 15584k reserved, 276k
data, 211k bss, 252k init)
Kernel virtual memory layout:
* 0xfff86000..0xfffff000 : fixmap
* 0xff800000..0xffc00000 : highmem PTEs
* 0xff7ec000..0xff800000 : early ioremap
* 0xe1000000..0xff7ec000 : vmalloc & ioremap
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Hierarchical RCU implementation.
NR_IRQS:512
mpic: Setting up MPIC " OpenPIC " version 1.2 at e0040000, max 1 CPUs
mpic: ISU size: 256, shift: 8, mask: ff
mpic: Initializing for 256 sources
clocksource: timebase mult[a00000] shift[22] registered
Console: colour dummy device 80x25
Mount-cache hash table entries: 512
Brought up 1 CPUs
NET: Registered protocol family 16
PCI: Probing PCI hardware
PCI: Cannot allocate resource region 0 of device 0000:00:00.0, will remap
pci 0000:00:00.0: PCI bridge, secondary bus 0000:01
pci 0000:00:00.0: IO window: 0x00-0xfff
pci 0000:00:00.0: MEM window: disabled
pci 0000:00:00.0: PREFETCH window: disabled
bio: create slab at 0
vgaarb: loaded
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
Freescale Elo / Elo Plus DMA driver
```

```
Switching to clocksource timebase
NET: Registered protocol family 2
IP route cache hash table entries: 4096 (order: 2, 16384 bytes)
TCP established hash table entries: 16384 (order: 5, 131072 bytes)
TCP bind hash table entries: 16384 (order: 5, 131072 bytes)
TCP: Hash tables configured (established 16384 bind 16384)
TCP reno registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
Trying to unpack rootfs image as initramfs...
Freeing initrd memory: 4497k freed
audit: initializing netlink socket (disabled)
type=2000 audit(0.512:1): initialized
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
NTFS driver 2.1.29 [Flags: R/O].
msgmni has been set to 1002
alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
Generic non-volatile memory driver v1.1
Serial: 8250/16550 driver, 2 ports, IRQ sharing enabled
serial8250.0: ttyS0 at MMIO 0xe0004500 (irq = 42) is a 16550A
console [ttyS0] enabled, bootconsole disabled
console [ttyS0] enabled, bootconsole disabled
brd: module loaded
loop: module loaded
nbd: registered device at major 43
st: Version 20081215, fixed bufsize 32768, s/g segs 256
Fixed MDIO Bus: probed
ucc_geth: QE UCC Gigabit Ethernet Controller
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
EDAC MC: Ver: 2.1.0 Feb 23 2014
Freescale(R) MPC85xx EDAC driver, (C) 2006 Montavista Software
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
Advanced Linux Sound Architecture Driver Version 1.0.21.
ALSA device list:
No soundcards found.
```

```
IPv4 over IPv4 tunneling driver
GRE over IPv4 tunneling driver
TCP cubic registered
Initializing XFRM netlink socket
NET: Registered protocol family 10
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
NET: Registered protocol family 15
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 252k init
ifconfig: SIOCSIFADDR: No such device
route: SIOCADDRT: No such process
Please press Enter to activate this console.
/ # gdb
/ # gdbserver [J
-/bin/sh: gdbserver: not found
/ #
```

As one can see it also faced problems configuring the loopback or perhaps binding the correct driver to the correct device? Or do we need to play with the networking option within QEMU as well?
I did try doing what you suggested however utilities like `file, ldd` and even `ld.so` seems to be missing.

```
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
NET: Registered protocol family 15
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 252k init
ifconfig: SIOCSIFADDR: No such device
route: SIOCADDRT: No such process
Please press Enter to activate this console.
/ # which gdbserver
/usr/bin/gdbserver
/ # ls -l /usr/bin/gdbserver
-rwxr-xr-x 1 1000 1000 878002 Mar 22 2014 /usr/bin/gdbserver
/ # file /usr/bin/gdbserver
-/bin/sh: file: not found
/ # ldd
-/bin/sh: ldd: not found
```

Is it because I compiled busybox as a static binary? Isn't it what busybox is by the way? So in case we need to run gdbserver as a standalone binary on a qemu system and not as a softlink to busybox, what exactly does one need to do?
Keen to hear from you.

Balau

2014/03/22

Yes you simply create the lib directory.
I think that compiling busybox without the "static" option might help. Busybox is a collection of many programs, but it doesn't have to be static. If it's static the shared libraries are compiled within, if it's not static the shared libraries are loaded from filesystem. If you have many programs that use the same shared libraries (like busybox and gdbserver) it saves disk space to have the shared libraries in the filesystem once, instead of compiled statically into the programs many times.
The fact that programs such as "file" are not present may be caused by some default options of Busybox, maybe they can be enabled to provide also "file" support.
So if you follow my blog post you will need to copy also the ld.*so.* (in my case ld-linux.so.3) of your toolchain in order to execute programs.
You can also try to simply run:

```
# /usr/bin/gdbserver
```

I don't know about the network issues, it probably depends on the machine you are emulating and on linux+busybox support for that machine.

6 Trackbacks For This Post

1. *Links 18/8/2010: PC-BSD 8.1 Reviewed, Vim 7.3 Released | Techrights* →
   August 18th, 2010 → 11:14
   […] Debugging ARM programs inside QEMU […]

2. *Links 20/8/2010: Google Chat Now on GNU/Linux, GNOME 2.32 Beta | Techrights* →
   August 20th, 2010 → 11:09
   […] Debugging ARM programs inside QEMU […]

3. *cat /proc/meminfo : MemTotal « Embedded Bits* →
   January 16th, 2011 → 21:58
   […] Whilst trying to understand this lot, I came across some very useful documentation on how the kernel manages it's memory. If you wish to read more try here, definitely here and some info on QEMU here. […]

4. *UK Embedded » Blog Archive » cat /proc/meminfo : MemTotal* →
   January 17th, 2011 → 04:22
   […] Whilst trying to understand this lot, I came across some very useful documentation on how the kernel manages it's memory. If you wish to read more try here, definitely here and some info on QEMU here. […]

5. *QEMU 관련 문서 | en-light-en-ment* →
   July 26th, 2012 → 12:33
   […] https://balau82.wordpress.com/2010/08/17/debugging-arm-programs-inside-qemu/ Share this:TwitterFacebookLike this:LikeBe the first to like this. […]

6. *QEMU 1.5.0 released, a backward compatibility warning | Balau* → May 21st, 2013 → 20:51
   […] Debugging ARM programs inside QEMU […]