QEMU + GDB Debugging Environment

18–349 Embedded Real-Time Systems

September 30, 2012

Contents

1 Introduction

This document explains how to set up the QEMU + GDB environment, and how to use it to improve the debugging experience for teams. QEMU[1] is a generic machine emulator that can run operating systems and programs built for one machine architecture on a different machine. In our case, we can emulate the ARM-based Gumstix platform on an x86/64–based laptop or desktop. QEMU also supports running a GDB server, an invaluable feature which allows us to debug kernel-mode code. Additionally, emulating the Gumstix platform can enable a team to be more productive, since team members can test their code without needing access to the limited Gumstix hardware.

2 Setting up the Software

Neither QEMU nor the ARM-version of GDB are installed on the CMU machines (`unix.andrew.cmu.edu`, `ece*.ece.cmu.edu`), so you must install them on your personal machine in order to use them. Although QEMU and GDB are available for Windows, Mac OS, and Linux, maintaining cross-compiling toolchains and cross-debugging environments is somewhat of a difficult, dark art. For this reason, we are going to describe how to set up the environment in Linux (specifically Ubuntu 12.04 LTS), using pre-packaged binaries whenever possible. If you don't already have a Linux development environment available, you can use VirtualBox to install and use Ubuntu 12.04 LTS, a popular Linux distribution.

2.1 Installing QEMU

The instructions here support installing QEMU version 1.0.50 on Ubuntu 12.04 LTS. If you would like to install and use QEMU + GDB on a Mac OS X system (or even a Windows system), the usage steps should be similar, but finding the exact packages for installation may be difficult. For Mac OS X, we recommend using the Homebrew package manager (though the packages may also be available through Fink or MacPorts).

As mentioned before, the easiest way to install QEMU is to use your favorite package manager to obtain the pre-packaged binaries. If you're feeling more adventurous, you can could compiling the latest version of QEMU from source. The packages needed for QEMU are:

1. `qemu` (base system which includes emulation support for x86/x86-64)
2. `qemu-system` (includes emulation support for other architectures, including ARM)

which can be installed with:

```
$ sudo apt-get install qemu qemu-system
```

After successfully installing QEMU, you should be able to run:

```
$ qemu-system-arm --version
QEMU emulator version 1.0.50 (Debian 1.0.50-2012.03-0ubuntu2)
```

2.2 Obtaining system files

In order to emulate the Gumstix platform in QEMU, we will need to obtain:

1. a Flash image (holds Gumstix bootloader, operating system)
2. an SD card image (can hold Gumstix operating system files, and our kernel and tasks)

[1] `http://wiki.qemu.org/Main_Page`

The flash image is the original contents of memory when the Gumstix boots. In theory, the flash is ROM. However, with QEMU this is not necessarily enforced, so you may have to either re-create the image periodically or re-download it if it gets corrupted (symptoms: your code may run wild). **When you download the files locally, make a copy before using QEMU (the images are large).**

You can download the two files here:

Flash image	`https://www.ece.cmu.edu/~ee349/f-2012/lab2/qemu/flash.img`
Alt. link	`http://www.contrib.andrew.cmu.edu/~acrichto/flash.img`
SD card image	`https://www.ece.cmu.edu/~ee349/f-2012/lab2/qemu/sdcard.img`
Alt. link	`http://www.contrib.andrew.cmu.edu/~acrichto/sdcard.img`

2.3 Using QEMU to emulate the Gumstix

Once you have all the necessary files, you can use this command to start emulating the Gumstix on your machine:

```
$ qemu-system-arm -nographic -M verdex -pflash flash.img -sd sdcard.img
```

Here you will see the familiar U-Boot prompt, and if you don't interrupt it, you should soon see the Gumstix login prompt.

2.4 Debugging QEMU with GDB

You can use GDB to debug code running on the QEMU-emulated Gumstix, which can be extremely helpful for debugging kernel-mode code. In order to do this, you need to start a GDB server when you first start emulating the Gumstix with QEMU. This can be done with the `-s` flag:

```
$ qemu-system-arm -nographic -M verdex -pflash flash.img -sd sdcard.img -s
```

However, before we can start debugging our ARM kernel-code, we need to install a version of GDB which runs on our x86/x86-64 based machine but can debug (or *target*) the ARM-based Gumstix (a *cross-debugger*). Again, we turn to our package manager and install the `gdb-multiarch` package:

```
$ sudo apt-get install gdb-multiarch
```

We can now connect to the Gumstix with:

```
$ gdb-multiarch
gdb> set architecture armv5te
gdb> target remote localhost:1234
```

At this point, QEMU should cease emulation because it's being controlled by GDB (it will stay stopped until you type `continue` in GDB).

We can also leverage symbolic debugging within GDB with just a few more commands (so you can type `break main` instead of `break 0xa292a3ec`). To support this, you first need to (re-)compile your code with debugging symbols enabled by adding the `-g` flag to the `CFLAGS` variable in your Makefile. Then, in GDB:

```
gdb> add-symbol-file path/to/your/kernel 0xa3000000
gdb> add-symbol-file path/to/your/user/binary 0xa000000
```

Congratulations, you can now inspect registers, memory, stack, and poke around symbolically within the comfort of a GDB environment!

3 Acknowledgements

This guide is an adapted version of Alex Crichton's (`acrichto`) original QEMU guide[2], presented to 18-349 in Fall 2011. It borrows heavily from Alex's instructions, and the course owes a big thank you to him for his cross-debugging research efforts. This instruction guide was adapted by Costas Akrivoulis (`cakrivou`).

[2] `http://www.contrib.andrew.cmu.edu/~acrichto/qemu.html`