Debugging with GDB

Workbook

Version v 1.0

October 2018

1 Debugging embedded devices using GDB

Objectives

Bugs happen. In this exercise you will have a chance run a debug session on the target board using gdbserver and GDB.

While you are running these labs, keep a copy of the GDB Quick Reference card open. It's in the file `gdb-refcard.pdf`.

1.1 Initial set up

At this point, you should have:

- A PocketBone card with BaconBits cape
- A micro SDcard containing `core-image-minimal-dev-pocketbeagle.wic.img`
- A copy of the Yocto Project SDK, `poky-glibc-x86_64-core-image-minimal-dev-cortexa8hf-neon-toolchain-2.5+snapshot.sh`

We need a working network connection between the target and host. For this we will use the USB RNDIS protocol.

There are a few things missing from the SDcard, so please download the tar ball from https://cm.e-ale.org/2018/debugging-ELCE-2018-csimmonds/update.tar.gz

Then, insert the micro SDcard into the SDcard reader and run the install script

```
$ ./update-sdcard.sh
```

When you boot the PocketBone again two network interfaces named **usb0** are created target and host. The target will be given address 192.168.7.2. For the host, you will need to configure a static IP address. The easiest way to do that is to edit `/etc/network/interfaces` and add these lines:

```
auto usb0
iface usb0 inet static
    address 192.168.7.1
    netmask 255.255.252.0
    network 192.168.7.0
    gateway 192.168.7.1
```

Now, if you haven't already, install the Yocto Project SDK by running the self-installing shell script:

```
$ cd <directory containing the e-ale downloads>
$ ./poky-glibc-x86_64-core-image-minimal-dev-cortexa8hf-neon-toolchain-2.5+snapshot.sh
poky-glibc-x86_64-core-image-minimal-cortexa8hf-neon-toolchain-2.5+snapshot.sh
```

Set up the shell environment to use the cross toolchain:

```
$ source /opt/poky/2.5+snapshot/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
```


Note: you need to do this for each new shell

Compile the sample helloworld program:

```
$ cd
$ cp -a <path to debug samples>/helloworld .
$ cd helloworld
$ $CC helloworld.c -o helloworld -ggdb
```

Verify that it has been cross compiled for an ARM target

```
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, inte
rpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=c1ff6dbe460c9ae4fea4121
80558497b8cbf459f, not stripped
```

Make sure that the micro SDcard is in the slot on the PocketBone and boot it up by inserting **both** USB cables into the PocketBone.

Run ifconfig on both host and target and check that the network is configured at both ends:

On the PocketBone serial console you should see this:

```
# ifconfig usb0
usb0      Link encap:Ethernet  HWaddr 22:AF:62:3F:DF:DF
          inet addr:192.168.7.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::20af:62ff:fe3f:dfdf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:115 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20692 (20.2 KiB)  TX bytes:1858 (1.8 KiB)
```

On your host laptop you should see this:

```
$ ifconfig usb0
usb0      Link encap:Ethernet  HWaddr 76:5e:91:4d:dc:95
          inet addr:192.168.7.1  Bcast:192.168.7.255  Mask:255.255.255.0
          inet6 addr: fe80::e428:ec49:f644:5edb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:116 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1592 (1.5 KB)  TX bytes:22511 (22.5 KB)
```

If the network is configured correctly, you can copy the helloworld program to the target:

```
$ scp helloworld root@192.168.7.2:/usr/bin
```

At this point you can use either (a) ssh to get a shell on the target:

```
$ ssh root@192.168.7.2
```

Or, (b) you can use the serial console (minicom)

Log on as root, no password, and run the helloworld program. You should see:

```
# helloworld
0 Hello world
1 Hello world
```



```
2 Hello world
3 Hello world
```

1.2 Debugging helloworld

The next task is to run helloworld in a GDB session.

On the target, launch the program with gdbserver:

```
# gdbserver :2001 /usr/bin/helloworld
Process /usr/bin/helloworld created; pid = 317
Listening on port 2001
```

Next, you need to find the sysroot the cross compiler. This must correspond to the directory containing the debug symbols that match the libraries that you are using on the target. **They must be the same, compiled from the same code base using the same cross compiler**

Generally, you can find the sysroot GCC uses with the option `-print-sysroot`. But with a toolchain from the Yocto Project SDK, you will see this:

```
$ arm-poky-linux-gnueabi-gcc -print-sysroot
/not/exist
```

In the case of Yocto Project, you read the sysroot from the compiler wrapper, CC:

```
$ echo $CC
arm-poky-linux-gnueabi-gcc -march=armv7-a -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a8i
--sysroot=/opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi
```

In this case the sysroot is
`/opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi`

If you are curious, you will find the debug symbols are in `lib/.debug` and `usr/lib/.debug` relative to the sysroot.

Now, on the host, launch the Yocto Project GDB:

```
$ cd helloworld
$ arm-poky-linux-gnueabi-gdb helloworld
GNU gdb (GDB) 8.2
[...]
Reading symbols from helloworld...done.
(gdb)
```

Now you can set the sysroot and connect GDB to the instance of gdbserver running on the target:

```
(gdb) set sysroot /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi
(gdb) target remote 192.168.7.2:2001
Remote debugging using 192.168.7.2:2001
Reading symbols from /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi/lib/
ld-linux-armhf.so.3...Reading symbols from /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-
poky-linux-gnueabi/lib/.debug/ld-2.28.so...done.
done.
0xb6fcea40 in _start ()
   from /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi/lib/ld-linux-armh
```

```
f.so.3
(gdb)
```

Set a breakpoint on `main` and continue...

```
(gdb) b main
Breakpoint 1 at 0x103de: file helloworld.c, line 7.
(gdb) c
Continuing.

Breakpoint 1, main (argc=1, argv=0xbefffe14) at helloworld.c:7
7               for (i = 0; i < 4; i++)
(gdb)
```

Run the program one step at a time using **next** instruction, which you can abbreviate to **n**.

When you come to the end of the program, quit GDB by typing **quit** which you can abbreviate to **q**

1.3 GDB command file

To reduce the amount of typing, create a GDB command file. Call it `helloworld-gdb.ini` and put these lines into it

```
set sysroot /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi
target remote 192.168.7.2:2001
```

Launch helloworld with gdbserver as before, then launch GDB like this

```
arm-poky-linux-gnueabi-gdb helloworld -x helloworld-gdb.ini
```

1.4 Looking at variables

Launch gdbserver running helloworld on the target as before. Launch gdb on the host, also as before, and continue to the main function.

Use the **print** command to display variable **i**:

```
(gdb) print i
```

Step through the program and see that i changes on each iteration.

Try setting i to a different number **just before** the printf:

```
(gdb)  set var i = 99
```

Note that the program prints out 99

Try setting **i** to a negative number

1.5 Shared libraries

To set breakpoints and step through library code you need to tell GDB where to find the source code.

GDB reads the executalbe to find the source directory, as you can see with this command:

```
$ arm-poky-linux-gnueabi-objdump --dwarf ./helloworld | grep DW_AT_comp_dir
```

Make a note of the directory for glibc.

In the case of the Yocto Project SDK, the source directories are in `/opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi/usr/src/debug`

So, you need to map from one to the other. The GDB command `set substitute-path` does that for you. For example:

```
(gdb) set substitute-path /usr/src/debug /opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnu
```

To try this out, use gdbserver to launch helloworld on the target again. On the host, run gdb as before, but this time add the `set substitute-path` command.

Now, when you get to the printf function, type **step** (abbreviated to **s**) to **step into** the function. This will take you into the source for the C library.

1.6 (Optional) JIT debugging

Let's take a look at what the init program is doing.

Use the attach option of gdbeserver to attach to init, which always has PID 1

```
# gdbserver :2001 --attach 1
Attached; pid = 1
Listening on port 2001
```

The path to init on the host is `/opt/poky/2.5+snapshot/sysroots/cortexa8hf-neon-poky-linux-gnueabi/sbin/init`

Launch gdb with that path

Set the sysroot and substitute-path as before, then attach to the target with `target remote`.

In gdb, type next. After a while init will wake up and run the next line of code. Now you have control of init and can debug it in the normal way.

Try the **backtrace** command to show the call stack to the current location.

Try stepping through init to see how it works.

1.7 (Optional) core dump

Copy ithe test program `may-crash` to the target and run it. It crashes!

```
# may-crash
0 Hello world
Segmentation fault
```

But, there is no core file, because the ulimit is not set.

```
# ulimit -c
0
```

So, go ahead and set the limit for core files to "unlimted":

```
# ulimit -c unlimited
```

Now run the program and it will generate a core file in the current directory.

This is usually inconvenient, so try creating a directory for core files and set a core pattern that references it:

```
# mkdir /corefiles
# echo "/corefiles/%e-%p" > /proc/sys/kernel/core_pattern
```

Run the program again - a core file is written to `/corefiles`

Copy the core file to your laptop and use GDB to look at the state of the program when is crashed.

```
$ arm-poky-linux-gnueabi-gdb <program name> <core file name>
```

