Debugging with GDB

Chris Simmonds

E-ALE 2018
These slides are available under a Creative Commons Attribution-ShareAlike 4.0 license. You can read the full text of the license here
http://creativecommons.org/licenses/by-sa/4.0/legalcode
You are free to

- copy, distribute, display, and perform the work
- make derivative works
- make commercial use of the work

Under the following conditions

- Attribution: you must give the original author credit
- Share Alike: if you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one (i.e. include this page exactly as it is)
- For any reuse or distribution, you must make clear to others the license terms of this work
About Chris Simmonds

- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at [http://2net.co.uk/](http://2net.co.uk/)

@2net_software

[https://uk.linkedin.com/in/chrisdsimmonds/](https://uk.linkedin.com/in/chrisdsimmonds/)
Objectives

• Show how to use GDB to debug applications
• How to attach to a running process
• How to look at core dumps
• Plus, we will look at graphical interfaces for GDB
• Reference: MELP2 Chapter 14

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it"
- Brian W. Kernighan
Native vs cross compiling

**Native** (on target)
- Makers, e.g. Raspberry Pi running Debian
- PC development

**Cross** (on host)
- Most embedded development
  - Better tools on dev host
  - Better integrated with SCM, etc
Remote debugging

Host

- Program symbols
  - gdb from toolchain

Target

- Program without symbols
  - gdbserver

Network or serial connection
Debug info

• Need debug info **on the host** for the applications and libraries you want to debug
  • It’s OK for the files on the target to be stripped: gdbserver does not use debug info

• Compile with
  • `-g`: for source-level debugging
  • `-g3`: to include information about macros as well

• Debug info may be included in the binary (the Buildroot way)

• Or placed in a sub-directory named `.debug/` (the Yocto Project/OpenEmbedded way)
Code optimization

- Single-stepping through optimized code can be confusing
  - Bad: -O2 and -Os
  - Bearable: -O1
  - Good: -Og (debug-friendly opt) or -O0 (no opt)
- If you experience problems, reduce the optimization level
- If back trace seems not to work, enable stack frames by adding to CFLAGS:
  - -fno-omit-frame-pointer
Setting sysroot

- **sysroot** tells GDB where to find library debug info

- For Buildroot
  
  ```
  set sysroot <toolchain sysroot>
  ```

- Using a Yocto Project SDK:
  
  ```
  set sysroot /opt/poky/<version>/sysroots/<architecture>
  ```
Command-line debugging

Development host

$ arm-poky-linux-gnueabi-gdb helloworld
(gdb) set sysroot /opt/poky/2.5.1/...
(gdb) target remote 192.168.7.2:2001

"Remote debugging from host 192.168.7.1"

(gdb) break main
(gdb) continue

{program runs to main()}

Embedded target

# gdbserver :2001 helloworld
Notes

- GDB command `target remote` links gdb to gdbserver
- Usually a TCP connection, but can be UDP or serial
- gdbserver loads the program into memory and halts at the first instruction
- You can’t use commands such as `step` or `next` until after the start of C code at `main()`
- `break main` followed by `continue` stops at `main()`, from which point you can single step
GDB command files

- At start-up GDB reads commands from
  - $HOME/.gdbinit
  - .gdbinit in current directory
  - Files named by gdb command line option `-x [file name]`
- Note: auto-load safe-path
  - Recent versions of GDB ignore .gdbinit unless you enable it in $HOME/.gdbinit

```
add-auto-load-safe-path /home/myname/myproject/.gdbinit
```
Library code

- By default GDB searches for source code in
  - $cdir: the compile directory (which is encoded in the ELF header)
  - $cwd: the current working directory

  (gdb) show dir
  Source directories searched: $cdir:$cwd

- You can extend the search path with the directory command:

  (gdb) dir /home/chris/src/mylib
  Source directories searched: /home/chris/src/mylib:$cdir:$cwd
Just-in-time debugging

• Both gdb and gdbserver can attach to a running process and debug it, you just need to know the PID

• With gdbserver, you attach like this (PID 999 is an example)

  # gdbserver --attach :2001 999

• If debugging natively using GDB, use the attach command:

  (gdb) attach 999

• In either case, to detach and allow the process to run freely again:

  (gdb) detach
A core file is created if:

- size is < RLIMIT_CORE
- the program has write permissions to create a file
- not running with set-user-ID
- Set RLIMIT_CORE to un-limited using command: `ulimit -c unlimited`
Using gdb to analyse a core dump

- Command-line gdb

```
arm-poky-linux-gnueabi-gdb sort-debug ~/rootdir/usr/bin/core
...
Core was generated by `~/sort-debug /etc/protocols'.
Program terminated with signal 11, Segmentation fault.
#0 0x00008570 in addtree (p=0x0, w=0xbeaf4c68 "Internet") at
sort-debug.c:45
  45     p->word = strdup (w);
(gdb) back
#0 0x00008570 in addtree (p=0x0, w=0xbeaf4c68 "Internet") at
sort-debug.c:45
#1 0x00008564 in main (argc=2, argv=0xbeaf4e34) at sort-debug.c:95
(gdb)
```
Core pattern

- By default, core files are called core and placed in the working directory of the program.
- If /proc/sys/kernel/core_uses_pid is non-zero a dot and PID number are appended.
- Or, core file names are constructed according to /proc/sys/kernel/core_pattern.
- See man core(5) for details.

Example: /corefiles/%e-%p

%e executable name
%p PID
• You need to add debug tools for the target: add this to your
  conf/local.conf

    EXTRA_IMAGE_FEATURES = "debug-tweaks tools-debug"

• And you need to build an SDK which will contain the tools for the host,
  and the debug symbols

    bitbake -c populate_sdk <image name>
• You need to run menuconfig and enable these options

```makefile
PACKAGE_HOST_GDB
PACKAGE_GDB
PACKAGE_GDB_SERVER
ENABLE_DEBUG
```

• Then re-build the image

• The executables with debug symbols are put in

```
output/host/usr/<arch>/sysroot
```
GUI front ends

- There are many front-ends, including
  - TUI: Terminal User Interface
  - DDD: Data Display Debugger
  - Eclipse CDT
- As an example, the next two slides show how to use DDD
DDD: Data Display Debugger

```
static struct twoidee *twoidee_toggle(twoidee *my, char w)
{
    int cond;
    if (p == NULL)
    {
        p = malloc(sizeof(struct twoidee *));
        p->data = strdup(w);
        p->left = NULL;
        p->right = NULL;
    } else if (cond == 0)
    {
        p->left = twoidee_toggle(p->left, w);
    } else
    {
        p->right = twoidee_toggle(p->right, w);
    } return p;

    /* Traversal the tree, printing out the words and counts */
    static void traversal(struct twoidee *my)
    {
        if (p is NULL)
        {
            (gdb) cont
            Continuing.
            Breakpoint 1, address (p) is 0xdeadbeef "services." at sort-debug.c:246
            (gdb) cont
            Continuing.
            Breakpoint 2, address (p) is 0xdeadbeef "Internet." at sort-debug.c:48
            (gdb) graph display (p->right) dependent on 1
            (gdb)
            In display 2: p->right word (double-click to dereference)
```
Starting DDD

- --debugger [GDB to use]
- -x [GDB command file]

Example:

$ ddd --debugger arm-poky-linux-gnueabi-gdb -x ~/gdbcmd [program]
Debugging kernel code

Outside the scope of this workshop, but ...

- Build kernel with KGDB - which is like gdbserver but integrated into the kernel
- Connect to serial port on target
- Read debug symbols from vmlinux file
Lab time...


Follow the notes in debugging-EALE-2018-csimmonds-workbook.pdf

Call me or one of the helpers if you encounter problems
Delving deeper

- This is an excerpt from my **Fast track to embedded Linux** class
- If you would like to discover more about the power of embedded Linux, visit [http://www.2net.co.uk/training.html](http://www.2net.co.uk/training.html) and enquire about training classes for your company
  - 2net training is available world-wide
- Also, my book, *Mastering Embedded Linux Programming*, covers the topics discussed here in much greater detail