Debugging in scratchbox

Lauri Arimo

Debugging in scratchbox

by Lauri Arimo

This is an document about debugging in scratchbox environment

Revision history

Version:	Author:	Description:
2004-11-08	Arimo	added some details
2004-09-30	Arimo	Initial revision

Table of Contents

Chapter 1. Introduction

1.1. Purpose of this document

This document introduces some tools that can be used for debugging and are provided with scratchbox. Focus is on describing differences between usage on native systems and scratchbox. This is not complete user manual for debugging tools, although some good reference pointers are given.

Reader is assumed to have good linux knowledge and some experience in debugging. Good place to start gathering common knowledge about debugging in linux environment is paper written by movial[1].

1.2. Scratchbox environment

Scratchbox[2] is a cross-compilation toolkit designed to make embedded Linux application development easier.

Due to the nature of scratchbox there can be binaries for different architectures. This mean that specially configured tools are needed for efficient debugging.

Chapter 2. Debugging

2.1. Debugging tools provided by scratchbox

Scratchbox provides following debugging tools by default:

- strace - A system call tracer[3]
- gdb - The GNU Debugger[4]
- gdbserver - Remote Server for the GNU Debugger[5]

2.2. strace

strace is a system call tracer, i.e. a debugging tool which prints out a trace of all the system calls made by a another process/program. The program to be traced need not be recompiled for this, so you can use it on binaries for which you don't have source.

With strace it is usually possible to get a grip what the problem is really fast. It is advisable to read strace man page and [3] to understand the power and the capabilities of the program. Strace will certainly make valuable addition to one's toolbox.

In scratchbox environment one usually wants to debug programs that are for different architecture. This is why strace has been compiled separately with every toolchain, so that one can always find strace that is compatible with current target. Precompiled strace can be found in scratchbox's device tools section, that will be something like: **/scratchbox/device_tools/strace-4.5.6/<insert toolchain name here>/bin/strace**

To use strace one must first copy it to the target file hierarchy so that it can be used with cpu-transparency. then it is just plain simple **/path/to/strace <arguments> program-to-be-run**. Strace will produce much output that can seem to be little cryptic at first glance, but one will soon learn to interpret it efficiently.

The thing to note here is the fact that strace binary to be used is compiled for the target architecture. This is why the execution will be done via cpu-transparency. This should not alter the result, it will only slow things down a bit.

Qemu should work quite well with strace. If this is not the case for some reason, one should try to use real cpu transparency device and sbrsh instead.

Strace can be used in two ways Either by running a new binary (e.g. **strace /bin/ls**) or by tracing existing process by specifying its process id (e.g. **strace -p 4352**). Other really useful flags are **-f** to follow forks, **-o** to write output to file and **-e** which can be used to filter output. Detailed information of can be found on strace man page and[3].

2.3. gdb

GDB is a source-level debugger, capable of breaking programs at any specific line, displaying variable values, and determining where errors occurred.

gdb needs to be configured separately for each toolchain. this is why it's built along sb's toolchains. user should not have to worry about this; scratchbox will put right version of the gdb to the PATH. If this is not the case, you have just found a bug. one can find correct version of gdb from **/scratchbox/compilers/$(COMPILER_NAME)/arch_tools/bin**.

If You plan to debug binary for some other architecture than x86, you need to use gdbserver to run the binary. gdbserver usage is described in more depth in the next chapter. To use gdbserver you need to give **target remote ip:port** command to gdb instead

To make debugging with gdb possible, one should compile binaries with **-g** flag so that debugging symbols are included in binary. It is possible to use stripped binary with gdbserver and tell to gdb where to look for debug symbols with **symbol-file /path/to/file** command. This is explained trough example at Example session chapter.

scratchbox's gdb has also been patched so that one should be able to read x86 and ARM core files on host machine. This is good way to find out what went wrong in the first place.

2.4. gdbserver

Gdbserver is a program that allows you to run GDB on a different machine than the one which is running the program being debugged. This means that one can run binaries for different architecture on target device with gdbserver via scratchbox's cputrancparency feature. Actual debugging can then be handled on host with gdb provided by scratchbox's toolchain.

Gdbserver may become handy because with it one can run binaries for different architecture on target device via cputrancparency feature of scratchbox. This is much faster than running whole gdb on target.

To use gdbserver one has to copy it first to the target filesystem. This is because gdbserver is binary for target architecture and this way it will be found in chrooted environment when using cputransparency.

gdbserver can be found at **/scratchbox/device_tools/gdb-6.1/<insert toolchain name here>/bin/gdbserver**

Launching gdbserver is easy; **gdbserver ip:port program.to.debug arguments.to.prog**. This will make gdbserver to listen debugging connections from ip at port port. It is also possible to attach gdbserver to already running process with **--attach pid** flag.

One has to use sbrsh as cpu-transparency method because qemu does not support gdb.

Currently gdbserver supports only following platforms:

- arm-*-linux-gnu
- i386-*-linux-gnu
- ia64-*-linux-gnu
- m68k-*-linux-gnu
- mips-*-linux-gnu
- powerpc-*-linux-gnu
- sh-*-linux-gnu

Chapter 3. Example session

3.1. general

In this chapter we have an example session of gdb and gdbserver usage in scratchbox. Chapter is divided to multiple sub-chapters so that it is easier to see whats going on. It is strongly advised to go trough this example to understand how gdb and gdbserver act together in scratchbox.

This chapter assumes that one has correctly setup scratchbox installation with working cpu-tranparency [6].

3.2. setup

First one needs some program to debug.

```
[sbox-ARM: ~/debug-test] > cat hello-world.c
#include <stdio.h>

int main(void)
{
        printf("Hello world!\n");
        return 0;
}
[sbox-ARM: ~/debug-test] > gcc -g hello-world.c
[sbox-ARM: ~/debug-test] > ./a.out
Hello world!
[sbox-ARM: ~/debug-test] >
```

3.3. gdbserver

First, copy gdbserver to target filesystem...

```
[sbox-ARM: ~/debug-test] > cp /scratchbox/device_tools/gdb-6.1/arm-linux-gcc-3.3_3.3.4-glib
[sbox-ARM: ~/debug-test] > file gdbserver

gdbserver: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.0.0, dynamicall
[sbox-ARM: ~/debug-test] >
```

and then just launch it with correct communication info and correct program with arguments

```
[sbox-ARM: ~/debug-test] > ./gdbserver 172.16.6.66:4444 ./a.out
Process ./a.out created; pid = 642
Listening on port 4444
```

Gdbserver is now waiting connections from 172.16.6.66 to port 4444. It will wait until interrupted or successful connection. After connection program execution goes as the gdb commands. Below is one possible output

```
[sbox-ARM: ~/debug-test] > ./gdbserver 172.16.6.66:4444 ./a.out
Process ./a.out created; pid = 642
Listening on port 4444
Remote debugging from host 172.16.6.66
Hello world!

Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
[sbox-ARM: ~/debug-test] >
```

quite simple, eh?

3.4. gdb

Check that correct gdb is in PATH

```
[sbox-ARM: /targets/links] > gdb -v
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-linux".
[sbox-ARM: /targets/links] >
```

Launch gdb, load correct symbols from correct file and connect to the gdbserver. Remember to use **cont** instead of **run** when starting the program. This is because program is already being ran by gdbserver.

```
[sbox-ARM: ~/debug-test] > gdb
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-linux".
(gdb) file ./a.out
Reading symbols from ./a.out...done.
(gdb) target remote 172.16.6.54:4444
```

```
Remote debugging using 172.16.6.54:4444
0x40000d00 in ?? ()
(gdb) cont
Continuing.

Program exited normally.
(gdb)
```

one can set breakpoints etc. before running program by **cont**. After the program exits, one has to restart gdbserver before new session.

3.5. gdb and corefiles

Scratchbox’s gdb can handle corefiles for x86 and arm targets.

Program must be compiled with debugging symbols (ie. **-g** switch) if some meaningful output is wanted.

```
[sbox-ARM: ~/debug-test] > cat segfault.c
int a (int *p);

int
main (void)
{
          int *p = 0;   /* null pointer */
            return a (p);
}

int
a (int *p)
{
          int y = *p;
            return y;
}
[sbox-ARM: ~/debug-test] > gcc -g segfault.c
[sbox-ARM: ~/debug-test] > ./a.out
[sbox-ARM: ~/debug-test] > gdb ./a.out core
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-linux"...

warning: core file may not match specified executable file.
Core was generated by ‘’.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
```

```
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x000083b0 in a (p=0x0) at segfault.c:13
13                int y = *p;
(gdb) bt
#0  0x000083b0 in a (p=0x0) at segfault.c:13
#1  0x0000838c in main () at segfault.c:7
(gdb)
```

Chapter 4. Graphical frontends

4.1. General

It’s possible to use some graphical frontend for scratchbox’s gdb. Perhaps the most popular of these is DDD - The Data Display Debugger, which can be used as frontend for many different debuggers.

The trick is to tell the frontend to use scratchbox’s gdb. In this document DDD is used as an example. Other frontends should work with same principles.

4.2. DDD

The Data Display Debugger (DDD) is a popular graphical user interface to UNIX debuggers such as GDB, DBX, XDB, JDB and others. Besides “usual” front-end features such as viewing source texts and breakpoints, DDD provides an interactive graphical data display, where data structures are displayed as graphs. Using DDD, you can reason about your application by watching its data, not just by viewing it execute lines of source code.

To launch DDD with scratchbox’s gdb use following command: **ddd --debugger ’/scratchbox/login gdb’**

More information about debugging with DDD can be found from [7].

References

[1] *Debugging linux applications (http://www.movial.fi/client-data/file/movial_debugging_linux_applications.pdf)* .

[2] *Scratchbox cross-compilation toolkit website (http://www.scratchbox.org/)* .

[3] *strace - A system call tracer (http://www.devchannel.org/article.pl?sid=03/10/24/2057246)* .

[4] *gdb - The GNU Debugger (http://sources.redhat.com/gdb/current/onlinedocs/gdb_toc.html)* .

[5] *gdbserver - Remote Server for the GNU Debugger (http://sources.redhat.com/gdb/current/onlinedocs/gdb_18.html#SEC146)* .

[6] *Installing Scratchbox (http://www.scratchbox.org/documentation/docbook/installdoc.html)* , Rahkonen Valtteri.

[7] *DDD - Data Display Debugger (http://www.gnu.org/software/ddd/manual/)* .