

Cross Platform Debugging for Raspberry

We will gdb on host and Pi to debug with the GCC toolset. First install the debug capability on our desktop system that works with target systems regardless of the processor architecture.

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

Now install a debug server on the Pi system that can be connected to from our desktop and control execution on the target.

```
sudo apt-get install gdbserver
```

Start the server on the Pi. The command starts the server, runs it continuously. The third parameter is the computer name or IP address of the host computer. In my case it is "192.168.178.18". 2001 is the port that the host and target will use for communication. Be aware that running the gdbserver is a potential security hole on the Pi. So make sure it can only be accessed from your local network.

```
gdbserver --multi 192.168.178.18:2001
```

The server uses stdout to display error messages and the output from the debugged program.

The program gdb-multiarch is an interactive program. On the host, enter the command:

```
gdb-multiarch
```

This prints a lot of output followed by the prompt (gdb). You might want to add the quiet option (-q) to the command line. At the prompt, using the name of your Pi, enter the following commands and you should see the responses, as shown in *italics* here:

```
target extended-remote 192.168.178.61:2001
Remote debugging using 192.168.178.61:2001
set remote exec-file remote/test
[[no response]]
file test
Reading symbols from test...done.
```

The target command tells gdb the name of the target system, 192.168.178.61 with port, 2001. If it worked, you'll get the response shown; otherwise a timeout message. The set remote line specifies the file location to use on the Pi target system. Remember that the executable needs to be copied onto the target system every time it is changed. We'll see how to do that from within gdb, below. The last command, file, is the path and filename on the host computer. This can be a directory other than where the debugger is running. Now, type the following and get the response as shown:

```
run
Starting program: /tmp/hello
[[ a bunch more lines, some warnings and stuff to ignore for now ]]
[Inferior 1 (process 234) exited normally]
```

If it works you'll see that last line. Now switch to the SSH terminal connected to the Pi and you'll see a couple of lines of status, the output from the program, a blank line, and a line reporting the exit status of the program. If all this doesn't work check the error message from the server. It should explain what is happening. Often there is a mismatch between what you told gdb and the location of the executable.

The debugger provides a large number of commands for manipulating breakpoints, continuing execution, listing the program, etc. For instance, type in main and the program stops when main() is reached. A 'c' runs the program after a breakpoint. To help with debugging, there is an interesting mode, the Terminal User Interface, or tui. You can access and leave it by typing in Ctrl-x, Ctrl-a. This mode provides windows that show the source, registers, assembly language, and other information.

Links

- <https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads>
- https://www.acmesystems.it/arm9_toolchain
- <https://jensd.be/1126/linux/cross-compiling-for-arm-or-aarch64-on-debian-or-ubuntu>
- <https://learn.arm.com/install-guides/gcc/cross/>
- <https://packages.debian.org/search?keywords=crossbuild-essential>
- <https://www.get-edi.io/assets/pdfs/DebianCross.pdf>
- <https://prashanth.entertolearn.in/debian-arm-architecture-and-emulation/using-debian-arm-cross-compiler-for-bare-metal-programming>

From:
<https://wiki.oscardegroot.nl/> - HomeWiki

Permanent link:
<https://wiki.oscardegroot.nl/doku.php?id=raspberry:cross-platform-compile:cross-platform-debugging&rev=1788507254>

Last update: 2026/09/04 07:34

