The Remote Debug Agent

The Remote Debug Agent

Table of Contents

Chapter 1. Using *RDA*, the *Red Hat Debug Agent*

User space applications may be debugged with GDB via a helper application called RDA. RDA is the Red Hat Debug Agent. It runs as a user space application on a board running the Linux kernel. RDA communicates with GDB over a TCP/IP network connection or serial connection using the GDB remote protocol. It translates protocol requests from GDB into the appropriate ptrace() operations and builds response packets from the results of the various ptrace() operations.

Prerequisites

Before using RDA, the following requirements must be met:

- The target must be running the Linux kernel with network support or support for some suitable serial device.
- The root file system running on the board must have a shell and the RDA application. GNUPro BSP root filesystems come with RDA by default.
- A network or serial interface must be available, configured, and visible to the host running GDB.
- The application to be debugged must be loaded onto some filesystem accessible from the board. It is permissible to strip debugging information from the application loaded onto the board.
- The application complete with debugging information must be accessible from the host.
- Shared libraries which are required by the application must be available on both the host and target via some means. It is quite helpful if the target's root filesystem is accessible on the host. If the target's root filesystem isn't available, then the toolchain's `sys-root` directory will often suffice.

The last requirement is important for debugging of applications which use shared libraries. A common problem is that the shared libraries are either not accessible from the host or are not exactly the same as those installed on the target. When installing updated shared libraries on the target, it is easy to forget to update the host's copies of these libraries.

Starting RDA for use with a network interface

When using a network interface for communication between RDA and GDB, RDA is started as follows:

rda [`-v`] ***`tcp-port-num`*** ***`executable-file`*** [***`arguments ...`***]

Consider the following command:

rda 1234 myapp

This will start RDA running at TCP port 1234. Once GDB connects, rda will attempt to start the application `myapp`.

The `-v` flag may be included on the RDA command line. Inclusion of this flag will cause RDA to print informational messages to `stderr`. These messages are occasionally useful to see if GDB is getting connected.

Any arguments supplied after the name of the executable file will become the arguments to the application started by RDA.

Starting RDA for use with a serial interface

When using a seral interface (such as an RS232 port) for communication between RDA and GDB, RDA is started as follows:

rda [`-v`] [`-s` speed] ***`device-name`*** ***`executable-file`*** **[*`arguments ...`*]**

Consider the following command:

rda -s 115200 /dev/ttyS0 myapp 42

This command will start RDA connected to serial port `/dev/ttyS0` running at 115200 baud. Once the serial port is open, RDA will attempt to start the application `myapp` whose single command line argument is `42`.

The `-v` flag may be included on the RDA command line. Inclusion of this flag will cause RDA to print informational messages to `stderr`. These messages are occasionally useful to see if GDB is getting connected.

If `device-name` is `-` (the dash character), RDA will connect to whatever device is attached to `stdin` and `stdout`. This makes it possible to start RDA from programs such as `inetd`.

Using RDA to attach to an existing process

RDA may be used to attach to and debug an already running process by use of the `-a` switch. The RDA command used to do this takes one of the following two forms (depending upon whether the transport mechanism is a network or serial interface).

Either:

rda -a [`-v`] ***`tcp-port-num`*** ***`process-id`***

Or:

rda -a [`-v`] [`-s` speed] ***`device-name`*** ***`process-id`***

In either case, `process-id` is the PID (process id) of the process to attach to. The PID is typically found using the **ps** command.

Obtaining RDA's usage message

The `-h` switch will cause RDA to display a brief usage message. For example:

rda -h

This command will cause RDA to display the following output:

```
Usage: rda [-v] tcp-port-num executable-file [arguments ...]
   or: rda -a [-v] tcp-port-num process-id
   or: rda [-v] [-s speed] device-name executable-file [arguments ...]
   or: rda -a [-v] [-s speed] device-name process-id
   or: rda -h
Start the Red Hat debug agent for use with a remote debugger.
Options and arguments:
  -a               Attach to already running process.
  -h               Print this usage message.
  -s speed         Set speed (e.g. 115200) at which device "device-name"
                   will communicate with remote debugger.
  -v               Increase verbosity.  One -v flag enables informational
                   messages.  Two -v flags turn on internal debugging
                   messages.
  arguments ...    Command line arguments with which to start program
                   being debugged.
  device-name      Name of serial device over which RDA will communicate
                   with remote debugger.
  executable-file  Name of program to debug.
```

```
process-id       Process ID (PID) of process to attach to.
tcp-port-num     Port number to which debugger connects for purpose
                 of communicating with the debug agent using the GDB
                 remote protocol.
```

Connecting to RDA with GDB

Prior to connecting with GDB, the executable's symbols must be loaded into GDB. This is either accomplished via GDB's **file** command or by simply specifying the name of the application to run on GDB's command line.

In order for GDB to find the symbols associated with shared libraries, it is essential that *solib-absolute-prefix* be set to the path on the host from which the target's `/lib` directory is available. (This should be the directory which contains `/lib`, not a the library directory itself). The current setting of `solib-absolute-prefix` can be displayed via the **show solib-absolute-prefix** command. If this prefix is not correct, it should be set via the **set solib-absolute-prefix** command. For example:

set solib-absolute-prefix /nfsroots/root-192.168.42.42

This will set `solib-absolute-prefix` so that the path `/nfsroots/root/root-192.168.42.42` will be prepended to absolute paths of libraries found by GDB's shared library machinery. If this path is not set correctly by default, it is often quite useful to add a suitable command to a `.gdbinit` file so that it will be performed automatically at startup time.

Once RDA is running, the GDB command **target remote** should be used to connect to the board running RDA. As noted earlier, either a network or serial interface must be configured and visible to the host running GDB. For example:

target remote 192.168.42.42:1234

Assuming that the board's network interface has been set to 192.168.42.42, GDB will attempt to connect to RDA at TCP port 1234. (If a hostname has been associated with this address via some sort of name resolution facility such as DNS, then the hostname may be used in place of the IP address.) Once connected, GDB will display something along the following lines:

```
Remote debugging using 192.168.42.42:1234
0xb75ebc50 in ?? ()
```

Alternately, if connecting to RDA using a serial interface, a slightly different **target remote** command is used. Consider the following GDB command:

target remote /dev/ttyS1

This command will attempt to connect to RDA over the serial port named `/dev/ttyS1`. Once connected, GDB will display a message much like this one:

```
Remote debugging using /dev/ttyS1
0xb75ebc50 in ?? ()
```

At this point, breakpoints may be set and the program may be run via GDB's **continue** command. (The **run** command is not used). See the GNUPro documentation for more information on using GDB.