NAME

gdbserver − Remote Server for the GNU Debugger

SYNOPSIS

gdbserver *comm prog* [*args*...]

gdbserver −−attach *comm pid*

gdbserver −−multi *comm*

DESCRIPTION

gdbserver is a program that allows you to run GDB on a different machine than the one which is running the program being debugged.

Usage (server (target) side):

First, you need to have a copy of the program you want to debug put onto the target system. The program can be stripped to save space if needed, as **gdbserver** doesn't care about symbols. All symbol handling is taken care of by the GDB running on the host system.

To use the server, you log on to the target system, and run the **gdbserver** program. You must tell it (a) how to communicate with GDB, (b) the name of your program, and (c) its arguments. The general syntax is:

```
target> gdbserver <comm> <program> [<args> ...]
```

For example, using a serial port, you might say:

```
target> gdbserver /dev/com1 emacs foo.txt
```

This tells **gdbserver** to debug emacs with an argument of foo.txt, and to communicate with GDB via */dev/com1*. **gdbserver** now waits patiently for the host GDB to communicate with it.

To use a TCP connection, you could say:

```
target> gdbserver host:2345 emacs foo.txt
```

This says pretty much the same thing as the last example, except that we are going to communicate with the `host` GDB via TCP. The `host:2345` argument means that we are expecting to see a TCP connection from `host` to local TCP port 2345. (Currently, the `host` part is ignored.) You can choose any number you want for the port number as long as it does not conflict with any existing TCP ports on the target system. This same port number must be used in the host GDBs `target remote` command, which will be described shortly. Note that if you chose a port number that conflicts with another service, **gdbserver** will print an error message and exit.

gdbserver can also attach to running programs. This is accomplished via the **−−attach** argument. The syntax is:

```
target> gdbserver --attach <comm> <pid>
```

pid is the process ID of a currently running process. It isn't necessary to point **gdbserver** at a binary for the running process.

To start `gdbserver` without supplying an initial command to run or process ID to attach, use the **−−multi** command line option. In such case you should connect using `target extended-remote` to start the program you want to debug.

```
target> gdbserver --multi <comm>
```

Usage (host side):

You need an unstripped copy of the target program on your host system, since GDB needs to examine its symbol tables and such. Start up GDB as you normally would, with the target program as the first argument. (You may need to use the **−−baud** option if the serial line is running at anything except 9600 baud.) That is `gdb TARGET-PROG`, or `gdb --baud BAUD TARGET-PROG`. After that, the only new command you need to know about is `target remote` (or `target extended-remote`). Its argument is either a device name (usually a serial device, like */dev/ttyb*), or a `HOST:PORT` descriptor. For example:

```
(gdb) target remote /dev/ttyb
```

communicates with the server via serial line */dev/ttyb*, and:

```
(gdb) target remote the-target:2345
```

communicates via a TCP connection to port 2345 on host 'the-target', where you previously started up **gdbserver** with the same port number. Note that for TCP connections, you must start up **gdbserver** prior to using the 'target remote' command, otherwise you may get an error that looks something like 'Connection refused'.

gdbserver can also debug multiple inferiors at once, described in the GDB manual in node `Inferiors and Programs` — shell command `info -f gdb -n 'Inferiors and Programs'`. In such case use the `extended-remote` GDB command variant:

```
(gdb) target extended-remote the-target:2345
```

The **gdbserver** option **--multi** may or may not be used in such case.

OPTIONS

There are three different modes for invoking **gdbserver**:

- Debug a specific program specified by its program name:

  ```
  gdbserver <comm> <prog> [<args>...]
  ```

  The *comm* parameter specifies how should the server communicate with GDB; it is either a device name (to use a serial line), a TCP port number (`:1234`), or `-` or `stdio` to use stdin/stdout of `gdbserver`. Specify the name of the program to debug in *prog*. Any remaining arguments will be passed to the program verbatim. When the program exits, GDB will close the connection, and `gdbserver` will exit.

- Debug a specific program by specifying the process ID of a running program:

  ```
  gdbserver --attach <comm> <pid>
  ```

  The *comm* parameter is as described above. Supply the process ID of a running program in *pid*; GDB will do everything else. Like with the previous mode, when the process *pid* exits, GDB will close the connection, and `gdbserver` will exit.

- Multi-process mode — debug more than one program/process:

  ```
  gdbserver --multi <comm>
  ```

  In this mode, GDB can instruct **gdbserver** which command(s) to run. Unlike the other 2 modes, GDB will not close the connection when a process being debugged exits, so you can debug several processes in the same session.

In each of the modes you may specify these options:

--help
List all options, with brief explanations.

--version
This option causes **gdbserver** to print its version number and exit.

--attach
gdbserver will attach to a running program. The syntax is:

```
target> gdbserver --attach <comm> <pid>
```

pid is the process ID of a currently running process. It isn't necessary to point **gdbserver** at a binary for the running process.

--multi
To start `gdbserver` without supplying an initial command to run or process ID to attach, use this command line option. Then you can connect using `target extended-remote` and start the

program you want to debug. The syntax is:

```
target> gdbserver --multi <comm>
```

--debug
Instruct `gdbserver` to display extra status information about the debugging process. This option is intended for `gdbserver` development and for bug reports to the developers.

--remote-debug
Instruct `gdbserver` to display remote protocol debug output. This option is intended for `gdbserver` development and for bug reports to the developers.

--debug-file=*filename*
Instruct `gdbserver` to send any debug output to the given *filename*. This option is intended for `gdbserver` development and for bug reports to the developers.

--debug-format=option1[**,option2,...**]
Instruct `gdbserver` to include extra information in each line of debugging output.

--wrapper
Specify a wrapper to launch programs for debugging. The option should be followed by the name of the wrapper, then any command-line arguments to pass to the wrapper, then `--` indicating the end of the wrapper arguments.

--once
By default, **gdbserver** keeps the listening TCP port open, so that additional connections are possible. However, if you start `gdbserver` with the **--once** option, it will stop listening for any further connection attempts after connecting to the first GDB session.

SEE ALSO

The full documentation for GDB is maintained as a Texinfo manual. If the `info` and `gdb` programs and GDB's Texinfo documentation are properly installed at your site, the command

```
info gdb
```

should give you access to the complete manual.

Using GDB: A Guide to the GNU Source-Level Debugger, Richard M. Stallman and Roland H. Pesch, July 1991.

COPYRIGHT