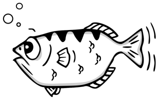
Valgrind and GDB integration

Using the GDB remote protocol to create a local interactive debugger experience for programs running under Valgrind

Mark Wielaard   Alexandra Hájková

September 21, 2023

Introduction



What are we going to talk about?

- How does Valgrind work?
- Valgrind (classic) gdbserver
- New GDB Valgrind python `monitor` commands
- `vgdb --multi` mode
- Remote protocol error handling
- GDB Remote Protocol extensions
- Handling I/O and terminals
- What is integrated on Fedora 38/39 (with Debuginfod)

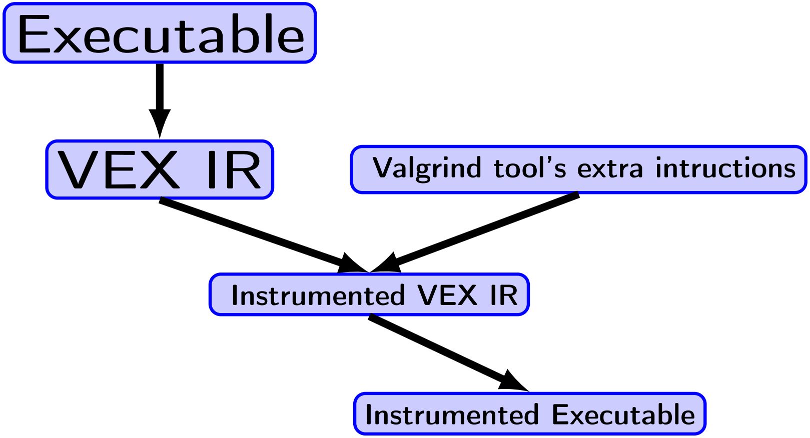
Valgrind

- an instrumentation framework for building dynamic analysis tools
- detects various memory management and threading bugs
- instruments your code
- intercepts syscalls, threading, auxv, /proc access
- not interactive `https://valgrind.org/`

Tools

- Memcheck
  - most used tool, default
  - detects unaccessible or undefined memory usage
- Cachegrind - cache profiler
- Massif - heap profiler
- Helgrind - thread debugger
- other tools

How Valgrind works



Valgrind's (classic) gdbserver

Valgrind's gdbserver

- 2011 fork from GDB's gdbserver
- Valgrind as remote target board to GDB
- vgdb - intermediary between GDB and Valgrind's gdbserver
  - to "wake up" Valgrind to talk to GDB

Connecting to Valgrind from GDB

- requires 2 terminal setup (GDB in one, Valgrind in another)
- provides "shadow" registers (XML target description)
- errors generate SIGTRAP
- various `monitor` commands

Valgrind specific `monitor` commands

`monitor` commands

- Sends special requests to gdbserver
- Just a blob of text
- Valgrind specific `monitor` commands
  - `monitor memcheck block_list`
  - `monitor memcheck leak_check`

Classic Valgrind/GDB Demo

First terminal

```
$ valgrind -q --vgdb-error=0 ./bad
==3781640== (action at startup) vgdb me ...
==3781640==
==3781640== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3781640== /path/to/gdb ./bad
==3781640== and then give GDB the following command
==3781640== target remote | /usr/local/lib/valgrind/../../bin/vgdb --pid=3781640
==3781640== --pid is optional if only one valgrind process is running
```

Second terminal

```
$ gdb ./bad

Reading symbols from ./bad...

(gdb) target remote | vgdb --pid=3781640
Remote debugging using | vgdb --pid=3781640
relaying data between gdb and process 3781640
warning: remote target does not support file transfer, attempting
to access files from local filesystem.
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/usr/lib64/ld-2.31.so.debug...
0x0000000004002110 in _start () from /lib64/ld-linux-x86-64.so.2
```

Classic Valgrind/GDB Demo

Second terminal

```
(gdb) break main
Breakpoint 1 at at 0x40120a: file bad_prog.c, line 30.
(gdb) continue
Continuing.

...

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00000000004011ed in setup_foo (s=0x1ffefff420) at bad_prog.c:23
23 s->buf[i] = malloc(20 * sizeof(int));
```

First terminal

```
==875162== Invalid write of size 8
==875162== at 0x4011ED: setup_foo (bad_prog.c:23)

...

==875162== (action on error) vgdb me ...
```

article

https://developers.redhat.com/articles/2021/11/01/debug-memory-errors-valgrind-and-gdb

Classic Valgrind/GDB Monitor Demo

evaluate command arguments manualy

```
(gdb) print &s.flag1
$4 = (int *) 0x1ffefff400
(gdb) print sizeof (s.flag1)
$6 = 4
(gdb) monitor xb 0x1ffefff400 4
ff ff ff ff
```

Valgrind memcheck monitor command

```
(gdb) monitor leak_check
==18002== 1,600 (+1,600) (1,440 (+1,440) direct, 160 (+160) indirect)
bytes in 18 (+18) blocks are definitely lost in loss record 3 of 3
==18002== at 0x4A36EA7: malloc (vg_replace_malloc.c:307)
==18002== by 0x4011EC: setup_foo (bad_prog.c:23)
==18002== by 0x401215: main (bad_prog.c:30)
==18002==
```

Python wrappers for valgrind monitor commands

- better integration in the GDB command line interface
- auto-completion, command specific help, searching for a command or command help matching a regexp, ...
- GDB will evaluate their arguments
  - `memcheck get_vbits &s.flag1`

Autoload the script

- Embed a `.gdb_script` section in `LD_PRELOAD` library
- Points to the script to autoload
- Script (not binary) but be in a "secure" location
- `--with-gdbscripts-dir=PATH` configure option
  - `--with-gdbscripts-dir=%{_datadir}/gdb/auto-load`

Use Help

```
(gdb) help valgrind

valgrind, v, vg

Front end GDB command for Valgrind gdbserver monitor commands.

Usage: valgrind VALGRIND\_MONITOR\_COMMAND [ARG...]

...

Type "help memcheck" or "help mc" for memcheck specific

Type "help helgrind" or "help hg" for helgrind specific

Type "help callgrind" or "help cg" for callgrind

Type "help massif" or "help ms" for massif specific
```

Help Memcheck

Help Memcheck

```
(gdb) help memcheck
memcheck, mc
Front end GDB command for Valgrind memcheck gdbserver monitor commands.
Usage: memcheck MEMCHECK_MONITOR_COMMAND [ARG...]
```

List commands

```
List of memcheck subcommands:
memcheck block_list -- Show the list of blocks for
memcheck check_memory -- Command to check memory .
memcheck get_vbits -- Print validity bits for LEN
memcheck leak_check -- Execute a memcheck leak search.
```

Monitor Demonstration

```
35          if (s.flag1 || s.flag2)
(gdb) memcheck get_vbits &s.flag1
ff
(gdb) memcheck get_vbits &s.flag2
00

(gdb) memcheck who_points_at &s.flag1
==777282== Searching for pointers to 0x1ffefffe02
==777282== tid 1 register RDI pointing at 0x1ffefffe02
```

Change Valgrind dynamic options

```
(gdb) valgrind v.clo
dynamically changeable options:
-v --verbose -q --quiet -d --stats --vgdb=no --vgdb=yes --vgdb=full
--vgdb-poll --vgdb-error --vgdb-stop-at --error-markers --show-error-list -s
--show-below-main --time-stamp --trace-children --child-silent-after-fork
```

- possible to change/add some options dynamically
    - `--quiet`
    - `--verbose`
    - `--trace-syscalls=yes`

GDB Remote Protocol

- communication between GDB and gdbserver/debugging stub
- GDB send commands, gdbserver/stub sends responses

target modes

- target **remote** mode
  - debugged program exits ⇒ GDB disconnects from the target
  - target decides what to run
- target **extended-remote** mode
  - debugged program exits ⇒ GDB remains connected to the target
  - GDB can ask to rerun or run a different program

Documentation

- `https://sourceware.org/gdb/onlinedocs/gdb/Packets.html`
- `https://sourceware.org/gdb/onlinedocs/gdb/General-Query-Packets.html`

`vgdb --multi` mode

- `vgdb --multi` allows to launch Valgrind from inside running GDB (also works with sockets `--port`)
- "traditional" target ⇒ extended-remote target
- allows rerunning program, keep breakpoints, settings, scripts, GDB history

vgdb

- sets up connection with GDB
- does early responses
- starts up Valgrind
- sets up connection with Valgrind gdbserver
- then just forwards packets

Extending Valgrind gdbserver to extended-remote

- Try to reuse as much as possible from existing valgrind gdbserver
- valgrind gdbserver doesn't need to know about `extended-remote` protocol
- Only `vgdb` implements `extended-remote` protocol
  - layer/shim on top of existing gdb target remote protocol
- No changes were made to the Valgrind gdbserver

Extended-remote protocol

- Packets implemented in `vgdb` for `--multi` mode
  - vMustReplyEmpty, qStartNoAckMode
  - '!', qSupported
  - qSetWorkingDir
  - qEnvironmentHexEncoded, qEnvironmentReset, qEnvironmentUnset
  - vRun
- Various other packets (reply with error or empty don't know)
  - qRcmd, qXfer, qAttached
  - qTStatus, qfThreadInfo
  - '?', 'H'

Extended-remote protocol `vRun` "handover"

- Capture status and replay on handover
  - NoAckMode
  - qSupported
  - cwd and environment (part of vRun)
- Implement `vRun`
  - `vRun;filename[;argument]...`
  - Run the program filename with the given arguments
    - `valgrind --vgdb-error=0 ...  <filename> <arguments...>`
  - filename could be empty - problem (no default)
  - it's not possible to report individual errors, like cwd failed
- valgrind gdbserver takes over
- vgdb just relays data without interpreting packets
- till valgrind ends, then vgdb starts interpreting packets again

Error handling

- should be improved
  - inconsistent
  - does not always reflect the documentation
  - Errors are numbers without meaning
- tried to improve, but more complicated than anticipated by backwards compatibility concerns.
- Solution might be something added (but never used) back in 2006

```
commit a76d924dffcb040b44a2bb5be026f0c974590c30
Author: Daniel Jacobowitz <drow@false.org>
Date:   Thu Sep 21 14:00:53 2006 +0000

* remote.c (packet_check_result): New function, split out
from packet_ok.  Recognize "E." as an error prefix.
```

Error handling proposal

- Document E.<error-string> variant.
- Check all error handling go through packet_check_result (which handles both E<hex><hex> and E.<error-string> variants)
- Make GDB gdbserver use E.<error-string> wherever possible (strerror)
- Do the same for the vgdb/valgrind gdbserver

Are you local?

- There were still some things the user had to setup to make GDB aware valgrind was running locally on the same machine
  - `$ gdb prog`
  - `(gdb) set remote exec-file prog`
  - `(gdb) set sysroot /`
  - `(gdb) target extended-remote | vgdb --multi`
- Could be a command or macro `target valgrind`
- But some protocol extensions could improve any local gdbserver
  - Default exec and arguments
  - Local machine/file system
  - Same environment and working directory

qDefaultExecAndArgs extension

- qDefaultExecAndArgs extension
  - Patch posted by Andrew Burgess
  - GDB asks gdbserver if there are default program and arguments
    - if there are none GDB will need to provide them with `vRun`
    - if they are set then GDB will remember them
  - Implementation for vgdb is trivial `send_packet ("U")`
    - No more need to `set remote exec-file`
  - Makes rerunning remote programs more consistent
    - `show remote exec-file` and `show args` will reflect how remote will be run

qMachineId extension

- qMachineId extension
  - Another patch posted by Andrew Burgess
  - Ask gdbserver for a machine description
  - qMachineId packet: `predicate;key=value[;key=value]*`
  - If descriptions match then GDB knows
    - it can safely ignore a 'target:' prefix in the sysroot
    - it can safely use the file specified with the 'file' command to start a remote inferior
  - Currently two attributes `bootid` and `cuserid`
  - Not yet implemented in vgdb (is bootid really unique?)

Environment and working directory

- At first we struggled with this, how to sync?
- But that was because to aid debugging we used sockets
    - `target extended-remote localhost:6666`
- Once we switched to “in process” target all this disappeared
- Still might be an extension to more easily sync setup with real remote?

Program input and output

- `target extended-remote | vgdb --multi` uses stdio
- Have to redirect stdin/stdout for inferior
- Propose protocol extension to switch file descriptors used for communication between GDB and gdbserver
- Real terminal handling for gdbserver/inferior

stdout/in redirection

- vgdb uses the same trick GDB gdbserver currently uses
- redirect `stdout` to `stderr`
- use `/dev/null` as stdin

How do we get stdin/stdout working?

```
/* When in stdio mode (talking to gdb through stdin/stdout, not
   through a socket), redirect stdout to stderr and close stdin
   for the inferior. That way at least some output can be seen,
   but there will be no input.  */
if (in_port <= 0) {
   /* close stdin */
   close (0);
   /* open /dev/null as new stdin */
   open ("/dev/null", O_RDONLY);
   /* redirect stdout as stderr */
   dup2 (2, 1);
}
```

Switch the file descriptors

Motivation

- give the inferior access to stdin

FdSwitch feature

- when gdbserver is run locally
- GDB preserves STDOUT/STDIN/STDERR file descriptors
- GDB sends preserved FD's to gdbserver
- gdbserver redirects its communication with GDB to sockets
- inferior is started connected to STDIN/OUT/ERR

Terminal handling

- mechanism for GDB to give a terminal control to the inferior

Motivation

- correct job control and signal delivery

Possible solutions

- terminal management hooks for the remote target
  - decide which job is currently the "foreground" job
  - just copied what existed in `inf_child_target`
  - only used with the FdSwitch feature
- having gdbserver pass all I/O over the remote protocol

Running Valgrind inside GDB Demonstration

```
$ gdb ./example
...
Reading symbols from ./example...
(gdb) set remote exec-file ./example
(gdb) set sysroot /
(gdb) target extended-remote | vgdb --multi --vargs -q
Remote debugging using | vgdb --multi --vargs -q


(gdb) start
Temporary breakpoint 1 at 0x4011a1: file example.c,
Starting program: /root/valgrind/example
relaying data between gdb and process 799017
Loaded /usr/share/gdb/auto-load/valgrind-monitor.py
Type "help valgrind" for more info.

Temporary breakpoint 1, main () at example.c:28
28      setup_foo(&s);
(gdb) c
Continuing.
==532003== Conditional jump or move depends on uninitialised value(s)
==532003==  at 0x401218: main (example.c:35)
==532003==
==532003== (action on error) vgdb me ...
Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000000000401218 in main () at example.c:35
35           if (s.flag1 || s.flag2)
(gdb) memcheck get_vbits &s.flag1
ff
(gdb) memcheck get_vbits &s.flag2
00
```

GDB with gdbserver all extensions applied

```
$ gdb /usr/bin/sort
GNU gdb (GDB) 14.0.50.20230907-git
Reading symbols from /usr/bin/sort...
Reading symbols from .gnu_debugdata for /usr/bin/sort...
(No debugging symbols found in .gnu_debugdata for /usr/bin/sort)
(gdb) target extended-remote | gdbserver --multi -
Remote debugging using | gdbserver --multi -
Remote debugging using stdio
(gdb) start
Temporary breakpoint 1 at 0x4088
Starting program: /usr/bin/sort
Process /usr/bin/sort created; pid = 3318599
Temporary breakpoint 1, 0x0000555555558088 in main ()
(gdb) c
Continuing.
GNU
GCC
^C
Program received signal SIGINT, Interrupt.
0x00007ffff79ff711 in __GI___libc_read (fd=0, buf=0x7ffff69e7018,
    nbytes=130048) at ../sysdeps/unix/sysv/linux/read.c:26
26          return SYSCALL_CANCEL (read, fd, buf, nbytes);
(gdb) c
Continuing.
GDB
^D
GCC
GDB
GNU
[Inferior 1 (process 3318599) exited normally]
(gdb) quit
Remote side has terminated connection.  GDBserver will reopen the connection.
```

Debuginfod

- GDB and Valgrind can use debuginfod out of the box (on Fedora 38)
- `DEBUGINFOD_URLS`
  - environment variable setup by default on Fedora 38
  - `debuginfod.fedoraproject.org`
  - `export DEBUGINFOD_VERBOSE=1`
  - `valgrind -v`
    - better way to see download information for Valgrind
  - GDB asks whether to use debuginfod
    - `Enable debuginfod for this session?  (y or [n])`
- an upcoming feature for both GDB and Valgrind
  - on-demand downloading of debuginfo - Aaron Merey

Summary

- `vgdb --multi`
  - valgrind 3.21.0, Fedora 38
- Python monitor commands
  - valgrind 3.21.0, Fedora 38
- stdout redirection
  - valgrind trunk, Fedora 39
- qDefaultExecAndArgs and qMachineId packet extensions
  - patches on gdb-patches
- FdSwitch packet support
  - `https://git.sr.ht/~sasshka/binutils-gdb/`
- gdbserver terminal handling
  - just a local hack and for GDB gdbserver, not vgdb yet
- debuginfod support
  - valgrind 3.21.0, Fedora 38
- lazy debuginfod loading
  - Valgrind trunk and Fedora 39

Contributors

- Andrew Burgess
  - "Improve GDB/gdbserver experience when using a local gdbserver" patch series
  - FdSwitch series consultations
- Philippe Waroquiers
  - The python monitor commands
  - Original valgrind gdbserver integration
- Aaron Merey
  - Implemented debuginfod support in Valgrind and GDB
- Mark Wielaard, Alexandra Hajkova
  - Extended remote vgdb “wrapper”
  - Integration of the above
- You!?
  - Ask questions
  - Make suggestions
  - Review code
  - Tell us what we got wrong