GDB Intro

Sergio Durigan Junior
sergiodj@{sergiodj.net,redhat.com,debian.org}
License

- License: Creative Commons Attribution 4.0 International License (CC-BY-4.0)
- https://creativecommons.org/licenses/by/4.0/
Introduction

➤ GDB: GNU project’s Debugger
Introduction

- **GDB**: GNU project’s Debugger (it is not a *database*...). Supports several programming languages.
- Started around 1986 by Richard Stallman (after **GNU Emacs**, but likely before **GCC**).
Compiling your program for GDB

- Your program needs to contain debug information (also called DWARF) for GDB to consume.

  # gcc -O0 -g program.c -o program, or
  # CFLAGS='-O0 -g' ./configure && make
Compiling your program for GDB

- Your program needs to contain debug information (also called DWARF) for GDB to consume.
- The GCC flag to include debug information is `-g`. We also use `-g3`, which includes information about macros (`#define`).

### Command Examples

- `# gcc -O0 -g program.c -o program`
- `# CFLAGS='-O0 -g' ./configure && make`
Compiling your program for GDB

- Your program needs to contain debug information (also called DWARF) for GDB to consume.
- The GCC flag to include debug information is `-g`. We also use `-g3`, which includes information about macros (`#define`).
- It’s common to disable optimizations when building the binary, by using the flag `-O0` (it’s `dash-oh-zero`).

Here's an example:

```bash
# gcc -O0 -g program.c -o program
```

Or

```bash
# CFLAGS='-O0 -g' ./configure && make
```
Your program needs to contain debug information (also called DWARF) for GDB to consume.

The GCC flag to include debug information is -g. We also use -g3, which includes information about macros (#define).

It’s common to disable optimizations when building the binary, by using the flag -00 (it’s dash-oh-zero).

# gcc -00 -g program.c -o program, or
# CFLAGS=''-00 -g’ ./configure && make
Running your program using GDB

- In GDB’s parlance, the program being debugged is called the inferior.
Running your program using GDB

- In GDB’s parlance, the program being debugged is called the inferior.

- Some ways to start the debugger:
Running your program using GDB

- In GDB’s parlance, the program being debugged is called the inferior.

- Some ways to start the debugger:
  - # gdb ./program
Running your program using GDB

- In GDB’s parlance, the program being debugged is called the inferior.
- Some ways to start the debugger:
  - # gdb ./program
  - # gdb --args ./program arg1 arg2
Running your program using GDB

- In GDB’s parlance, the program being debugged is called the inferior.

- Some ways to start the debugger:
  - `# gdb ./program`
  - `# gdb --args ./program arg1 arg2`
  - `# gdb`
    (gdb) file ./program
    (gdb) run arg1 arg2
    Or you can also use `start` (run and stop at main).
{Break,Catch,Watch}points

- A **breakpoint** is related to *source code* (location, function). A **watchpoint** is related to *data* (read/write of a variable). A **catchpoint** is related to an *event* (enter/exit a syscall, fork, receive a signal).
A breakpoint is related to source code (location, function). A watchpoint is related to data (read/write of a variable). A catchpoint is related to an event (enter/exit a syscall, fork, receive a signal).

Breakpoints (code)
- break
- tbreak (temporary)
Breakpoints (code)
- break
- tbreak (temporary)

Watchpoints (data)
- watch (write), rwatch (read), awatch (access)
- Conditional watchpoints are supported.
{Break, Catch, Watch}points

- A breakpoint is related to source code (location, function). A watchpoint is related to data (read/write of a variable). A catchpoint is related to an event (enter/exit a syscall, fork, receive a signal).

- Breakpoints (code)
  - break
  - tbreak (temporary)

- Watchpoints (data)
  - watch (write), rwatch (read), awatch (access)
  - Conditional watchpoints are supported.

- Catchpoints (events)
  - catch fork
  - catch syscall
Resuming the execution

▶ After GDB has stopped the inferior (because a *point has been hit, for example), you will probably want to resume its execution.
Resuming the execution

- After GDB has stopped the inferior (because a *point has been hit, for example), you will probably want to resume its execution.

- You may just want to continue the program:
  - continue
Resuming the execution

▶ After GDB has stopped the inferior (because a *point has been hit, for example), you will probably want to resume its execution.

▶ You may just want to continue the program:
  ▶ continue

▶ Or maybe go to the next statement/instruction:
  ▶ next (*statement*), or nexti (*instruction*)
Resuming the execution

- After GDB has stopped the inferior (because a *point has been hit, for example), you will probably want to resume its execution.

- You may just want to continue the program:
  - continue

- Or maybe go to the next statement/instruction:
  - next (statement), or nexti (instruction)

- Or step into a function:
  - step (statement), or stepi (instruction)
Resuming the execution

- After GDB has stopped the inferior (because a *point has been hit, for example), you will probably want to resume its execution.

- You may just want to continue the program:
  - continue

- Or maybe go to the next statement/instruction:
  - next (statement), or nexti (instruction)

- Or step into a function:
  - step (statement), or steipi (instruction)

- Or finish executing the current function, but stop at the end:
  - finish
Examining data

▶ The inferior has stopped... Now what?
Examining data

- The inferior has stopped... Now what?
- You may want to print the value of some variable:
  - `print VAR`
Examining data

- The inferior has stopped... Now what?
- You may want to print the value of some variable:
  - print VAR
- Or examine a memory location:
  - x ADDRESS
Examining data

- The inferior has stopped... Now what?
- You may want to print the value of some variable:
  - print VAR
- Or examine a memory location:
  - x ADDRESS
- The type of a variable? Easy:
  - whatis VARIABLE
Examining data

- The inferior has stopped... Now what?
- You may want to print the value of some variable:
  - `print VAR`
- Or examine a memory location:
  - `x ADDRESS`
- The type of a variable? Easy:
  - `whatis VARIABLE`
- Hint: you may want to enable pretty-printing:
  - `set print pretty on`
Examing the code

▶ Yes, we have ncurses! The Text User Interface!
  ▶ C-x a (that’s CTRL x a).
Examining the code

- Yes, we have ncurses! The Text User Interface!
  - C-x a (that’s CTRL x a).
- If you want to list the current region, or if you don’t want/can’t to use TUI:
  - list
Examining the code

▶ Yes, we have ncurses! The Text User Interface!
  ▶ C-x a (that's CTRL x a).

▶ If you want to list the current region, or if you don’t want/can’t to use TUI:
  ▶ list

▶ You can also disassemble code:
  ▶ disassemble
Examining the code

- Yes, we have ncurses! The Text User Interface!
  - C-x a (that’s CTRL x a).

- If you want to list the current region, or if you don’t want/can’t to use TUI:
  - list

- You can also disassemble code:
  - disassemble

- If GDB can’t find the source code, you can specify its location using the dir command.
Examining the call stack

▶ If you want to see the call stack (A.K.A. stack trace) that lead to the current function:
  ▶ bt
Examining the call stack

- If you want to see the call stack (A.K.A. stack trace) that lead to the current function:
  - `bt`

- And you can move through it:
  - `up` and `down`
  - You can also go to a specific frame: `frame NUMBER`
Corefiles

- Corefiles are frozen images of the inferior. You can inspect everything that was happening when the process was running (but you can’t resurrect it).
Corefiles

- Corefiles are frozen images of the inferior. You can inspect everything that was happening when the process was running (but you can’t resurrect it).

- You can generate them outside GDB, when a program crashes. Make sure you:
  - `ulimit -c unlimited`
  - Check if systemd is handling them
    (/proc/sys/kernel/core_pattern).
Corefiles

- Corefiles are frozen images of the inferior. You can inspect everything that was happening when the process was running (but you can’t resurrect it).

- You can generate them outside GDB, when a program crashes. Make sure you:
  - ulimit -c unlimited
  - Check if systemd is handling them (/proc/sys/kernel/core_pattern).

- You can also generate them inside GDB, at any moment:
  - generate-core-file
Corefiles are frozen images of the inferior. You can inspect everything that was happening when the process was running (but you can’t resurrect it).

- You can generate them outside GDB, when a program crashes. Make sure you:
  - ulimit -c unlimited
  - Check if systemd is handling them (/proc/sys/kernel/core_pattern).

- You can also generate them inside GDB, at any moment:
  - generate-core-file

- You can open a corefile using GDB:
  - # gdb program -c corefile.PID
Other interesting information

▶ info breakpoints
▶ info locals
▶ info registers
▶ Many others!
Who you gonna call?

- Our online documentation (info) is very good!
- Every command has a help.
- You can also use `apropos` when searching for a term.
- TAB-completion is also useful.
Other advanced features

- Python support.
- Reverse debugging.
- Support for SystemTap SDT probes.
Thank you

- Thanks to Red Hat for the support.
- Thanks to Paul Nijjar and Bob Jonkman for the invitation.
- Thanks to you for watching!