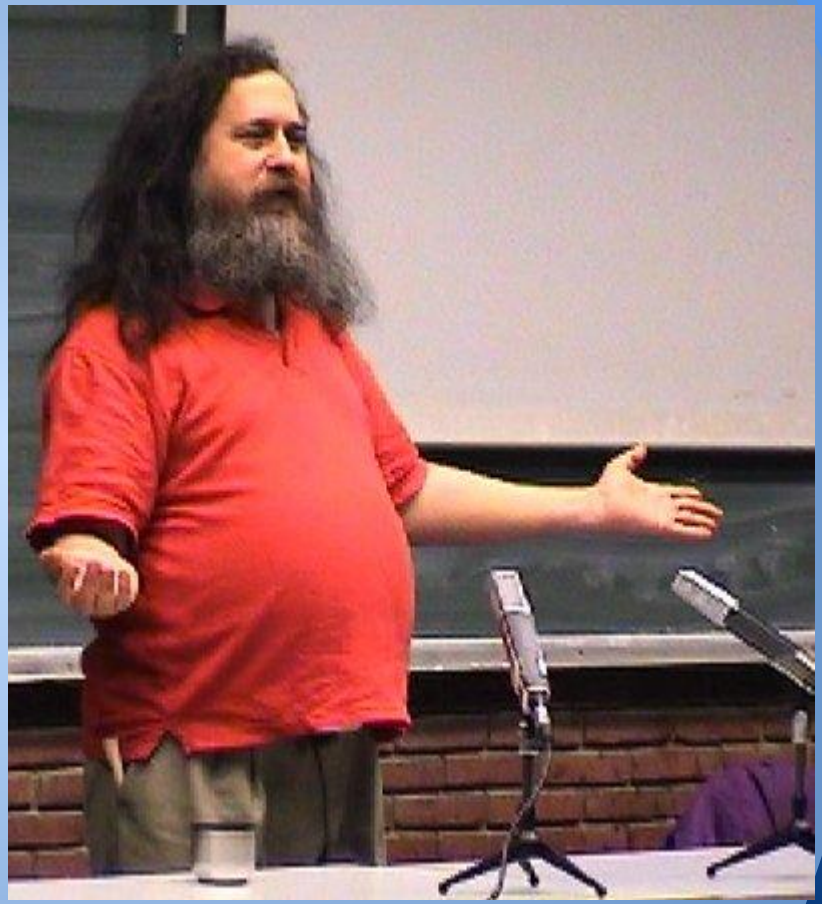
How2GDB

September 28, 2014

Phil Lopreiato, Neel Shah

What's a GDB?

- GNU Debugger
- Written by Richard Stallman in 1986
- Currently maintained by Free Software Foundation



How do I get it?

- Windows
  - Enable ‘gdb’ in Cygwin setup
  - Use MinGW for gcc toolchain (http://www.mingw.org/)
- OSX/Linux
  - Look harder.
  - lldb in Yosemite (Similar thing, but Apple’s version. -- If you have Yosemite, see Neel)

How do I use it?

- Helps you debug programs when they aren’t working properly
  - Incorrect values
  - Segfaults
  - It’s all broken
  - When you’ve run out of tears

Let’s get started…

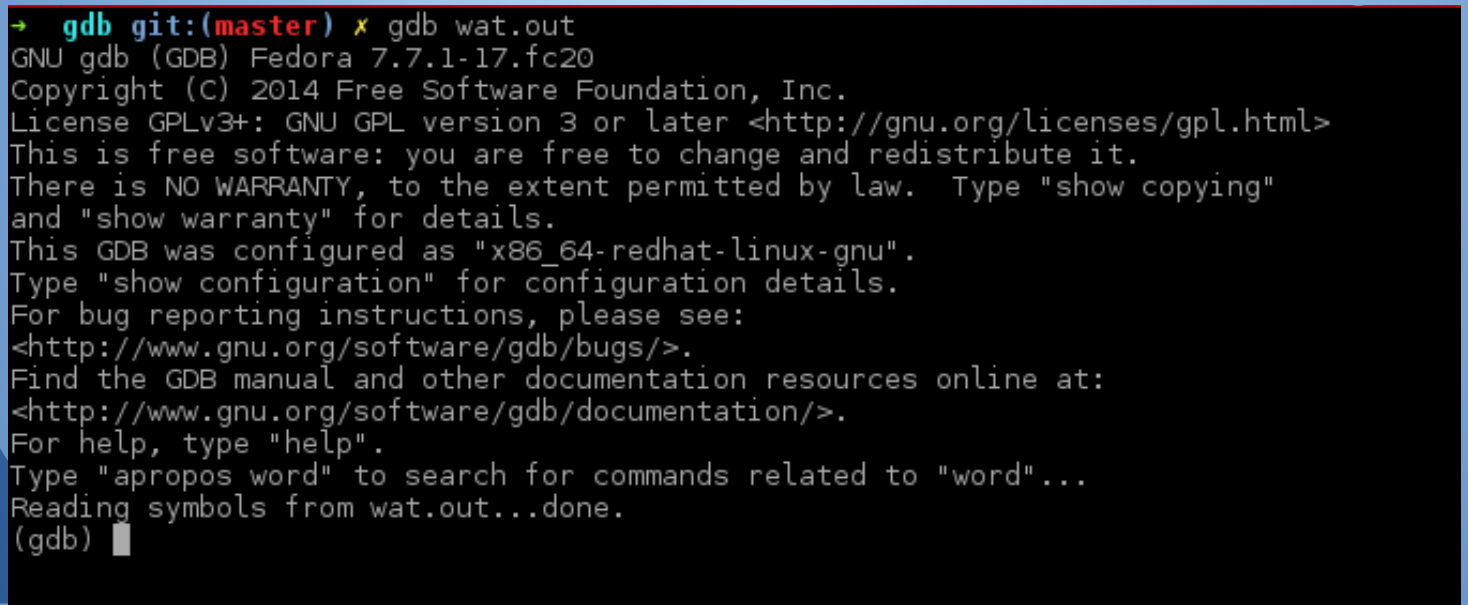
- Get code from: shah7.com/gdb

Step 1

- Compile your code with debugging symbols enabled
  - Add -g flag to gcc command
  - Example: gcc -g -o wat wat.c

Step 2

- Run the program with GDB
  - gdb wat



The GDB Prompt

- Run ‘help’ for a list of commands

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

Step 3

- Run the program through GDB
  - run

```
→  gdb-workshop git:(master) ✗ gdb wat
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/v0id/ACM/gdb-workshop/wat...done.
(gdb) run
Starting program: /home/v0id/ACM/gdb-workshop/wat

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400518 in main () at segfaults.c:7
7           printf("Our string is: %s\n", *string);
(gdb) :^[
```

Investigating Segfaults

- Use the ‘backtrace’ command to see the function calls prior to the segfault

```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040053c in main () at segfaults.c:7
7           printf("Our string is: %s\n", *string);
(gdb) backtrace
#0  0x000000000040053c in main () at segfaults.c:7
(gdb)
```

- Use the ‘list’ command to see the code surrounding the error

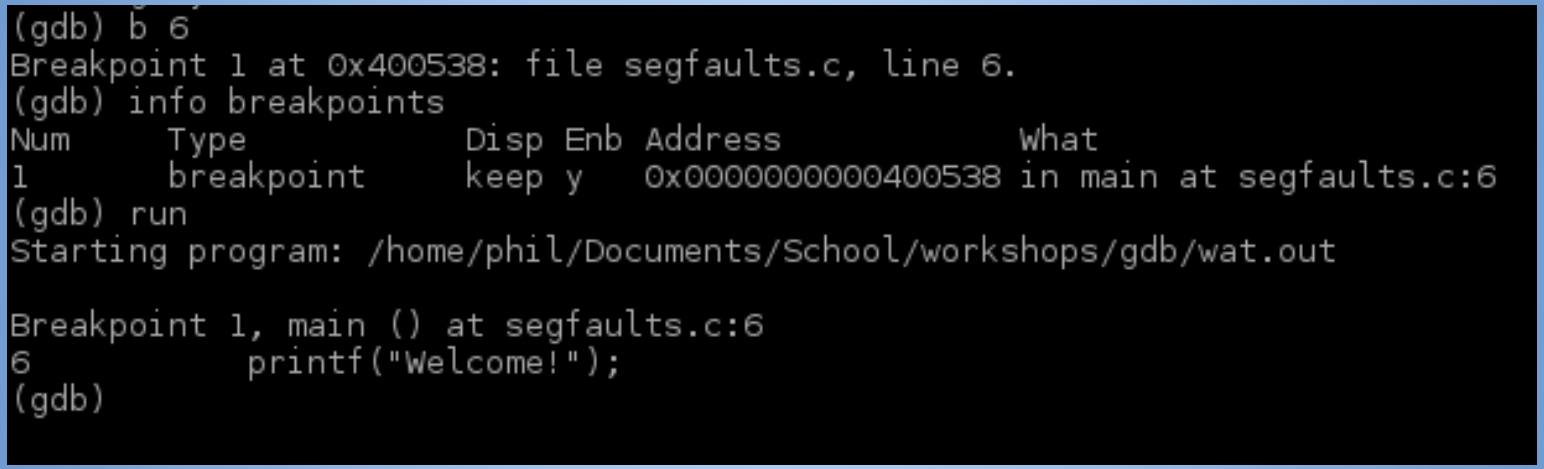
```
(gdb) list
2
3       int
4       main(void)
5       {
6           char *string;
7           printf("Our string is: %s\n", *string);
8           *string = 'W';
9           printf("Animals make: %s\n", string);
10
11      }
(gdb)
```

Breakpoints

- Breakpoints are a way of telling C to stop the program at a certain point
- You can examine memory once the program is stopped at a point
  - Check contents of variables at a point in your code
- Great for debugging!

Breakpoint Commands

- Set breakpoint
  - break <line num>
- Set breakpoint on function
  - break <func name>
- List of breakpoints
  - info breakpoints
- Remove breakpoints
  - clear <breakpoint num> ← retrieved from info
- Skip breakpoints
  - ignore <breakpoint num> ← retrieved from info


(gdb) b 6
Breakpoint 1 at 0x400538: file segfaults.c, line 6.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000000400538 in main at segfaults.c:6
(gdb) run
Starting program: /home/phil/Documents/School/workshops/gdb/wat.out
Breakpoint 1, main () at segfaults.c:6
6 printf("Welcome!");
(gdb)

Step through code line-by-line

- When execution is stopped at a breakpoint, use ‘step’ to execute ***just*** the next line

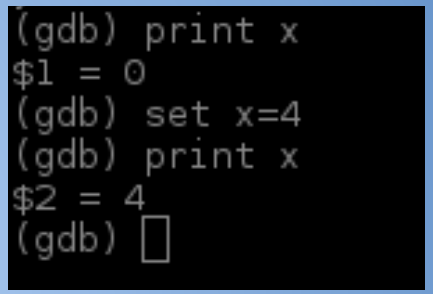
Exercise: Fix segfaults.c

- Using gdb and your knowledge of C, modify the segfaults.c program so that it runs without error.
  - Common commands:
    - `run`
    - `b <line>`
    - `list`
    - `step`
    - `continue`
    - `quit`

Inspecting Your Code

- gdb also allows you to view and modify all kinds of data used by the program such as
  - variables
  - stack frames
  - memory

View/Modify Variables



- Display variable contents
  - print <var name>
- Modify variable contents
  - set <var name> = <data>
- Watchpoints - stop and notify on variable change
  - watch <var name>
- Disable watchpoint
  - get id from ‘info breakpoints’
  - disable <id>

Stack Frames

- The ‘backtrace’ command returns current active stack f

```
(gdb) backtrace
#0  __find_specmb (format=<optimized out>) at printf-parse.h:108
#1  _IO_vfprintf_internal (s=0x38cb1b8400 <_IO_2_1_stdout_>, format=0x4006e0 "Bitwise inspection p
    at vfprintf.c:1308
#2  0x00000038cae51b69 in __printf (format=<optimized out>) at printf.c:33
#3  0x00000000004005e8 in main () at bitwise.c:16
(gdb)
```

- Use ‘info frame’ to see details about the currently active frame

```
(gdb) info frame
Stack level 0, frame at 0x7fffffffdaf0:
 rip = 0x38cae47ad8 in __find_specmb (printf-parse.h:108); saved rip = 0x38cae51b69
 inlined into frame 1
 source language c.
 Arglist at unknown address.
 Locals at unknown address, Previous frame's sp in rsp
(gdb)
```

More on Stack Frames

- Use ‘info args’ to see the arguments into the current frame
- Use ‘info locals’ to see the frame’s local variables
- Use frame <n> to change stack frames

```
(gdb) frame 3
#3  0x00000000004005e8 in main () at bitwise.c:16
16          printf("Bitwise inspection program!\nEnter one number: ");
(gdb) info args
No arguments.
(gdb) info locals
x = 0
y = 0
(gdb)
```

View/Modify Memory

- You can examine memory with the x command
  - x/FMT_ADDRESS
    - char *s = “Hello world\n”
      - (gdb) x/s s ← examine variable as string
      - (gdb) x/c s ← examine variable as character
      - (gdb) x/4c s ← examine variable as 4 characters
      - (gdb) x/t s ← examine first 32 bits of variable
      - (gdb) x/3x s ← examine first 24 bytes of variable in hex
    - FMT_ADDRESS has a lot of combinations - full list via ‘help x’ in GDB

Exercise: What does it do?

- Using gdb's abilities to access, modify, and watch variables, what does the function in bitwise.c do?
- Remember, you can use:
  - `print`
  - `set`
  - `watch`
  - `backtrace`
  - `frame`