GDB Tutorial

Prof. Bo Hong

ECE 2035 Fall 2013

Last modified September 11th, 2014

What is GDB?

- GNU Debugger
- Works for several languages
  - including C/C++
- http://www.sourceware.org/gdb/documentation/

Not Every Executable Is Debugging-Friendly

- Two conditions need to be met:
  1. The executable has to carry *debugging symbols*
     - The symbols tell GDB where to look in the source
     - To include symbols, use the – g flag for gcc
       - E.g.: gcc hello.c –g –o hello
  2. The source tree needs to remain the same as it was when the program was compiled
     - You cannot move the source to a different directory and expect GDB to *magically* know this

Starting GDB

- Use command: “gdb <executable>”
- Output will look like:

```
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-45.el5)
Copyright (C) 2009 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /nethome/bhong6/gatech_teaching/fall13/2035/C/hello...done.
(gdb)
```

Starting GDB

- If gdb was started without the executable parameter

```
bash$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-45.el5)
Copyright (C) 2009 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) file hello
Reading symbols from /nethome/bhong6/gatech_teaching/fall13/2035/C/hello...done.
(gdb)
```

The GDB environment

- It has tab completion!

  (gdb) dis (followed by tab key)
  disable disassemble disconnect display

- It remembers your commands
- "Enter" repeats the previous command
- Not sure about a command?

  (gdb) help command

- It supports short forms!

  help -> h
  break -> b
  …

Showing the source

- The list command (or “l” for short)

```
(gdb) list
1       #include <stdio.h>
2       #include <stdlib.h>
3
4       int main(void) {
5               // comment
6               int sum;
7               int i;
8               for(i=0;i<10;i++) {
9                       sum += i;
10              }
```

- By default, lists 10 lines, but can specify location and amount:

```
(gdb) l 23        (lists 10 lines centered around 23)
(gdb) l 5, 25     (lists lines 5 to 25)
```

Running

```
(gdb) l main, 20
4       int main(void) {
5               // comment
6               int sum;
7               int i;
8               for(i=0;i<10;i++) {
9                       sum += i;
10              }
11              printf("%d\n", sum);
12              return 0;
13      }
```

(gdb) run

```
Starting program: /nethome/bhong6/gatech_teaching/fall13/2035/C/hello
45

Program exited normally.
(gdb)
```

What if there are bugs?

- Step through the code a bit at a time
- Stop at an error
- Examine the memory/code
- Find out what happened

- New gdb commands are needed

Breakpoints

- Set it

```
(gdb) break cstd.c:8
Breakpoint 1 at 0x4004a0: file cstd.c, line 8.
```

- Run program and stop at it

```
(gdb) run
Starting program: /nethome/bhong6/gatech_teaching/fall13/2035/C/hello
Breakpoint 1, main () at cstd.c:8
8               for(i=0;i<10;i++) {
```

- Check around

```
(gdb) print i
$2 = 0
(gdb) print sum
$3 = 0
```

Breakpoints

- You can have as many breakpoints as necessary, program will stop if it reaches any of them
- You can break at a line, or a function, or an address

```
(gdb) break 8
Breakpoint 5 at 0x4004a0: file cstd.c, line 8.
(gdb) break cstd.c:8
Breakpoint 6 at 0x4004a0: file cstd.c, line 8.
(gdb) break main
Breakpoint 7 at 0x4004a0: file cstd.c, line 8.
(gdb) break *0x4004a0
Breakpoint 8 at 0x4004a0: file cstd.c, line 8.
(gdb)
```

So you stopped at a break point

- Investigate, using the print command

```
(gdb) print i
$4 = 0
(gdb) print sum
$5 = 0
(gdb) print /x sum
$6 = 0x0
```

Now you investigated, what's next?

- Continue until next breakpoint or program completion

  (gdb) continue

- Single-step – forward by one line of source code

  (gdb) step

- Next-line – forward, treats a function as one line

  (gdb) next

- Stop the program

  (gdb) kill

How about showing something when stepping along?

(gdb) display variable_name

```
(gdb) display i
5: i = 4
(gdb) display sum
6: sum = 6
(gdb) next
8               for(i=0;i<10;i++) {
6: sum = 10
5: i = 4
(gdb) next
9                       sum += i;
6: sum = 10
5: i = 5
(gdb)
```

How about breaking automatically when some variable is changed?

(gdb) watch variable_name

How about adding condition to a breakpoint?

(gdb) condition breakpoint_num what_cond

```
(gdb) break 9
Breakpoint 1 at 0x4004a9: file cstd.c, line 9.
(gdb) condition 1 i==4
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004004a9 in main at cstd.c:9
        stop only if i==4
(gdb) run
Breakpoint 1, main () at cstd.c:9
9                   sum += i;
6: sum = 0
5: i = 4
(gdb)
```

Other Useful Commands

```
(gdb) delete 1        //remove breakpoint 1
(gdb) info break      //show information about all breakpoints
(gdb) undisplay 1     //un-display variable 1
(gdb) finish          //run till the end of current function
(gdb) Ctrl-x a        //toggle text-user-interface (TUI) mode
(gdb) Ctrl-x 1        //force TUI to use one window
(gdb) info stack      //show the stack information, useful, will discuss later
```