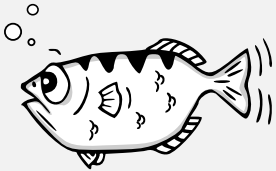

An introductory tutorial on GDB

CS240 – Data Structures

What is GDB?

GNU Debugger:

- Free and open source tool
- Lots of programming languages supported
- Debug your buggy code!
- Find out where your program crashes
- Pause execution before crash point.
- Examine/alter variables (prevent the bug!)

Setup and Usage

- GDB is already available on the department computers
- Must use `-g` flag when compiling your code (applies to *gcc* for C, *g++* for C++, *javac* for Java)
- Run with

  ```
  gdb <executable>
  ```

- Or with

  ```
  gdb --args <executable> <executable arguments>
  ```

Functionality

gdb accepts commands from an interactive prompt

- Your program has not began executing yet
- The debugger is waiting for your command
- You should see something like this:

```
jmal@tartaros ~/gdb_tutorial % gdb segfault
GNU gdb (Debian 7.7.1+dfsg-5) 7.7.1
Copyright (C) 2014 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 "i586-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from segfault...done.
(gdb)
```

Basic Commands

Starting and Stopping

- **start**: Begin execution
- **quit(q)**: Stop execution and quit debugger

Execution Flow Control

- **break(b)**: Instruct debugger to stop execution on specified point
  1. One of your function names
  2. **<File>:<Line>**: Specified line on specified file
- **delete(d) <Num>**: Remove breakpoint <Num> (or all breakpoints with no argument)
- **next(n)**: Execute current (source language, not assembly!) instruction and go to next
- **continue(c)**: Execute until hitting a breakpoint or the program finishes/crashes

Analyzing Runtime Info

We now know how to navigate our program

How do we actually inspect variables and function calls?

- **print(p)**: Our debugging swiss army knife
  1. Print any kind of variable
  2. **print x**: Display the current value for variable x
  3. Expressions in your source code language supported
  4. For example, **print &x** valid for a C program
  5. Can also change variable values, **print x=1** actually changes x
- **backtrace(bt)**: Display function call stack (with function arguments)
  - Useful to understand the behaviour of our code and whether bad things can happen
- **frame(f)**: Display stack frame for current function

Segfault Hunting Guidelines

How to deal with segfaults?

3 step Process

1. Find the segfaulting code
2. Understand why it segfaults
3. Apply a fix

Finding the segfault

To quickly find the origin of a segfault

1. Launch the debugger and set no breakpoints
2. Enter **start** and **continue** after the automatic breakpoint
3. Note the reported filename and line associated with the segfault

Understanding the segfault

This is the hardest step!

A segfault is an attempt to access memory which not ours. Look at the instruction causing the segfault

- Ask yourself: which variable in the instruction could reference a memory address?
- –> A pointer to a structure?
- –> An array position?

Run the debugger again:

1. Add a **breakpoint** to the function which causes the segfault
2. **start** and **continue** to reach the breakpoint
3. Now go step by step using **next**
4. Check often the values of suspected variables using **print**

Fixing the segfault

The solution to the segfault is heavily linked to the conditions which created it in the first place.

Understand how your guilty variable got its illegal value

- Was it provided by the function arguments?
- Was it the result of a missing check ? (most often)
- Was it the result of another function's bad implementation? (most difficult to debug, use **backtrace**)

Once all that is done

You are probably not done!

- There will (probably) be more bugs
- The larger the code base, the more bugs
- And the harder it is to debug

More info about GDB, debugging

We only covered a tiny subset of the capabilities of GDB

- Use command **help** in gdb for a list of commands
- Use **help <command>** for more info on a specific command
- GDB homepage
- GDB documentation
- MIT tips on segfaults