Basic Debugging With GDB

Benjamin S. Skrainka
University College London
Centre for Microdata Methods and Practice

April 5, 2015

What is GDB? Why do I care?

GDB is an open source debugger:

- Works with C, C++, and FORTRAN
- Interfaces:
  - Command line
  - GUI available (Eclipse, jEdit, etc.)
- Find errors in your code much more quickly than `printf`
- Watch how your code executes:
  - Run to a breakpoint (or you crash)
  - Walk through code one line at a time
- Post-mortem after a crash
- Once you know one debugger you know them all....

Overview

This talk will teach you the basics skills needed to use a debugger:

1. Building your code for the debugger
2. Running your code
3. Navigating through your code
4. Examining what is going on
5. Advanced features

Building your code for debugging

Design your code to be debugged:

- Do not use printf or equivalent
  - Slows development down because you must keep recompiling every time you want to look at a new variable
  - Slows code down
  - Makes code more difficult to understand
  - Voluminous output hard to track
  - Must remove printf once your code is working....
- Some diagnostic logging is sensible, but there is no need to ‘roll your own’:
  - http://log4c.sourceforge.net/
  - Google Logger glog
- Can use the macro trick to optionally enable/disable diagnostics

Debugging

Use the C preprocessor to facilitate debugging (even in FORTRAN):

```
#ifdef USE_DIAG
#define DIAG_PRINT      PRINT *,
#else
#define DIAG_PRINT      !
#endif
```

Defensive Programming

Practice defensive programming:

- Defensive programming:
  - Choose a sensible design
  - Separate application into separate libraries/modules
  - Access all resources via a library
  - Helps you track down a bug
- Write unit tests to exercise your code as early as possible in the development cycle
- The sooner you catch a bug the less time it takes to fix
- Get your coding working first, then optimize (using gprof)

Houston, we have a problem...

You wrote your code but it fails. Now what?

- Remain calm
- Diagram the system
- Explain the problem/code to someone
- Change one thing at a time
- Keep an audit log
- Divide and conquer to find smallest reproducible case
- Did it work before you made a change?
- Add logging
- List possible causes of the error
- Something you think is true isn't

See *Debugging* by David J. Agans

Compile for Debugging

Compile your code for debugging:

- GDB needs extra symbol information
- Enable with −g compiler flag
- Works best without optimization so use -O0 -fno-inline as well
- Slower than production code with full compiler optimization enabled ... but you can debug it

Starting the Debugger

To start GDB, invoke it from the command line:

```
bss$ gdb GDBTest
GNU gdb 6.3.50-20050815 (Apple version gdb-1510) (Wed S
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public
welcome to change it and/or distribute copies of it und
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show wa
This GDB was configured as "x86_64-apple-darwin"...Read
(gdb)
(gdb) quit
```

Documentation

There are many resources for help:

- (gdb) help *command*
- Google
- GDB online documentation

Breakpoints

Before running your application, you must set a breakpoint:

- When you start your code, GDB will run until it hits a breakpoint or your application crashes
- Set with the `break` command:

```
break fooLib.c
break foolib.c:666
break foofunc
break 123
break main
```

- Can customize break points so they are conditional, etc.
- May need to display code to point GDB to the correct file using `list` *`file[:lineNum]`* | *`lineNum`* | *`func`*

Manipulating Breakpoints

Some common breakpoint commands:

`info break`	list breakpoints
`disable n`	disable breakpoint *n* temporarily
`enable n`	enable breakpoint *n*
`delete n`	delete breakpoint *n*
`delete`	delete all breakpoints

Running your code

To run you code, use the run command – do not forget to specify the command line arguments

```
(gdb) break main
Breakpoint 1 at 0x100000d1b: file GDBTest.c, line 52.
(gdb) run 5
Starting program: /Users/bss/sbox/docs/teaching/BasicSk
Breakpoint 1, main (argc=2, argv=0x7fff5fbff408) at GDB
52    nStatus = GetArgs( argc, argv, &nFac ) ;
```

Navigating through your code

There are four basic commands for moving through an application:

`step`	move to next line, enter functions
`next`	move to next line, skip over function calls
`continue`	run to next breakpoint or crash
`finish`	complete execution of current function call

Examining what is going on

The basic commands are:

`info args`	information about function arguments
`info locals`	information about automatic variables
`info reg`	information about registers
`bt`	display call stack (could use info stack)
`p` **VarName**	print *VarName*
`x` **/fmt address**	examine memory at *address* and display using format *fmt*
`p` **fooFunc()**	executes and prints return value of *fooFunc()* (could use call *fooFunc()*)
`display` **VarName**	print *VarName* every time execution stops

- Note: you may need to dereference pointers...
- There are many `info` commands for examining how your program is running

The Call Stack

Every time a function is called a new frame is pushed on the stack.
To find a bug, you will may need to examine it:

`bt`	print call stack
`where`	print call stack
`up`	move up one stack frame
`down`	move down one stack frame
`frame` *n*	go to frame *n*
`info frame`	information about current frame

Advanced features

GDB has many additional features:

- Abbreviate commands by using just the first couple letters of a command, e.g. `i b`
- Modify variables or GDB's state using `set`
- `source` *`File`* runs all the commands in *File* as if you typed them in
- Customization:
  - Specify start up commands in `.gdbinit` file
  - Write your own commands
- Other user interfaces to GDB exist: emacs, cgdb, Eclipse