Lesson 5

GDB: The GNU Project Debugger

GDB

The GNU Project Debugger

Debuger – Why ?

Simple way to debug – textDebugging

Just print out what you are inspecting

For more complex task you can use syslog

Don’t forget to remove it before production :)

Debuger – Why ?

While text debuging is simple, it can lead to a lot of unreadable text, morever, it can take long time to get the real bug, viewing a few varibales at a time.

Also, there is no way to Pause your app, and look around, that may be very helpfull.

Debuger – What ?

Debug – a Tool (or event in a small environment) that can controll an execution of your code.

GDB can do a lot of usefull thigs like:
BreakPoints, StackTrace, Variable viewing, Examinating core dumps, and more..

BreakPoint may be done in many ways. By function, by line, by thread,
and even **by condition**

Debuger – How ?

If your app crashes, you probably want to get an idea, why, and when this happens.

For this, we need a “core dump”. A snapshot of the environment fot the moment of crash.

To enable it, run: *ulimit -c unlimited*

Learn by example !

Debuger – How ?

Once we have run our simplePrint and it fails, try to understand why this happens.
lets see the core dump by:

gdb -c <coreFile> <executable>

```
Core was generated by `./simplePrint'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000400548 in print ()
(gdb)
```

Debuger – How ?

```
Core was generated by `./simplePrint'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000400548 in print ()
(gdb)
```

SIGSEGV (SIGNAL 11) it’s not too informative by itslf, but means wrong memory access (bad pointer, array out of bounds..)

to get where it happens, compile with debug symbols

gcc -o simplePrint -g3 simplePrint.c

Debuger – Commands

bt -backtrace – shows the way to function

fr <num> can change strack frame

List – show the code around crash.

List can move by lines to showe code above or after: list +/-10

List <num> will take you to line number.

Debuger – Commands

breakpoint: there is may options with this command. f.ex.
break 12 - will set breakpoint to line 12
break print – will set breakpoint to function “print”

Info breakpoints – show information about what bp set, and if they have been reached

Delete – remove break points

Debuger – flow control

r – run : will run our debugeble programm
c – continue: continue after break point
ctrl+c : pause execution
kill – wil stop current execution
q – quit gdb

connect to running process:
sudo gdb <./executable_name> -pid <pid>

Debuger – get value of varible

To print a value:

p – print <var_name>

p/d will convert the output do decimal

help all – display a lot of options

Debuger – How to change a value

Connect to a process or run localy

List local variables by “info local” or “bt full”

By “bt” and “fr” navigate to loop function, and get value of i

By “set variable i=5” change the vaule of i

By “continue” let the process run, and check what happens

Contidional Break: break if i == 80