Lesson 1 – Libs, GDB

Introduction

- Who am I ?
- Contacts
- Why this course ?

- Who am I ?
  Aharon (Arkady) Gorodischer
  About 10 yeras of developer experience
- Contacts
  Email: arkadyg@ariel.ac.il

- Why this course ?
  OS is your running enviroment, and should be efectivly used by a software developer.
  It may help you with a lot of already made tools, but may become your nightmare, if you are not aware of how it should be handeled

Libs

Some times we need to devide our code for separate files. Moreover , we need to devide it for reusable parts, and client specific.

For this, the Library was envented.
A stand-alone, already compiled (and probably tested) bunch of code, for developers to be used.

Libs

There is at least two Libraries types:
Static and Dynamic (also called Shared)

Static usualy ends with “.a” .
In compile time, they are integrated in the executable file.

Dynamic ends with “.so” (shared object).
In compile time, they are linked to the code, but not included in executable, and can be replaced

Libs - shared

Name convention:
Lib name will be lib<name>.so
f.ex libCore.so, libAlgoImageTracking.so

Compilation of the lib:
gcc -o libHello.so -shared -fPIC hello_ariel.c

Libs - shared

Compilation of code:
gcc main1_1.c -L. -l Hello -o hello2

-L measn where to look for the library

-l (small L) measn whats the name of the library.
note that the “lib” and “.so” are omited

export LD_LIBRARY_PATH=.
updates LD, where to look for the lib in run time

Makefile in 5 minutes

Propably should include an “all” task, and “clean”

The structure is:
task name : dependency (files or tasks)
(tab) cmd command (gcc...)

.PHONY: clean “header” will become before a command, independed of the filesystem (otherwise,if you will have file named “clean” it will make a bug)

Libs - tools

How can we realy know what is in our library and what it made for ?

file – shows some basic info. Is it executable or shared lib, for that platform (x86,arm) was build

nm – shows that symbols are inside. Our functions shoud be listed too. (note that c++ will change funtions name)

Libs - executable

How can we know that our executable depends on libraries ?

ldd – will display the dependencies of an executable.

ltrace – will display library calls

strace – will display system calls

Libs – dynamic loading

Can we load a library without compiling against it?

dlopen – load the library (lasy – when needed)

dlsym – looks for a symbol (function name)

Example !

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 !

(check /var/lib/apport/coredump/)

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