GDB Essentials

Advanced Operating Systems

Overview

- Coding practices for the labs

- GDB
  - Setting up
  - Walkthrough
  - Frequently used gdb commands

- Useful git commands for the labs

Coding etiquette

1. Use comments - not too few, not too many
   /* `Going stingy can hurt later!` */

2. Follow existing coding style.

3. Keep the infra intact
   - Avoid tinkering with makefiles
   - Use `conf/env.mk` to set compile flags
4. Sane `git commit` messages

Coding etiquette

3. Keep the APIs intact

```
int page_alloc(int alloc_flags)
{
    ...
}
```

```
int page_alloc(int alloc_flags,
               size_t size)
{
   ...
}
```
✗

```
int buddy_alloc(int alloc_flags,
               size_t size);

int page_alloc(int alloc_flags)
{
  return buddy_alloc(alloc_flags,
                     4096);
}
```
✓

5. Compile with & w/o -DBONUS_LAB<*n*>

Debugging

- Execution flow + Incorrect behavior
- Program binary file
- Symbols
- Source code

```
$ gcc -O0 -g <program.c> -o <program>

$ gcc -O0 -g3 <program.c> -o <program>
```

gdb - setup

- gdb initialization scripts
  - ~/.gdbinit
  - .gdbinit
  - -x <init script>

- `set history save`

gdb - example

```
$ gcc -O0 -g -o ln *.c
```

```
$ ./ln -sr “” /tmp/symlink
```

Download files from:

http://goo.gl/ItVEIW

Or http://tinyurl.com/gdb-ln

gdb - example

```
$ ./ln -sr “” /tmp/symlink
```

What is going wrong?

gdb - spotting the fault

- **`backtrace`**

  Print backtrace of all stack frames

- **`frame [n]`**

  Select and print a stack frame

  frame 0 → current frame

  frame 1 → caller’s frame

gdb - exploring the state

- `info <registers|stack|variables|...>`
  - `registers`
  - `proc mappings`
  - `variables`
  - `breakpoint`

to inspect target’s state

- `x/FMT <address>` (*eg: x/100xw $esp*)
- `display/i` OR `x/i`
- `print <value>`
- `printf <fmt>, <var list>`

gdb - walking the path

- ❑ **break**
- ❑ **watch**

- ❑ **continue**
- ❑ **step [n]**
- ❑ **next**
- ❑ **stepi / si**

★ Conditional breakpoint

```
break 155 if realfrom == NULL
```

★ `commands <break num>`

```
> your list of
> gdb commands
> end
```

gdb - exploring the state

- `disassemble`
- `symbol-file <filepath>`
- `list`

to explore program's code

- `set  args | history |...`
- `show args | history |...`

debugger settings

gdb - exploring the state

- ❑ **`define <function_name>`**

```
> your list of
> gdb commands
> end
```

- ❑ **`layout asm`**
- ❑ **`layout split`**
- ❑ **`layout regs`**

Allows keeping an eye on code, registers, etc, while you step through the execution.
Use Ctrl+X A to switch back

gdb

Many many more…

- `apropos <search term>`
- `help <command>`

Debugging OpenLSD

```
$ make qemu[-nox]-gdb
   qemu-system-i386 -gdb tcp::<port> -S
```

```
$ make gdb
   gdb -x .gdbrc
```

Enabling symbols:

In `conf/env.mk`, add:

```
CFLAGS=-O0 -g3
```

Debugging OpenLSD

- Triple fault → Then what?

- a `breakpoint` just before it faults?
  EIP value would help

Git

```
$ git clone
https://github.com/vusec/aos-labs.git

$ git remote add <devpt> <your git repo>

$ git commit -am <commit message1>
$ git commit -am <commit message2>
$ …
$ git push devpt <branch>
```

Getting the next lab's framework

```
$ git pull --rebase
$ git fetch --all --tags --prune
```

Rebase new lab on top of your changes:

```
$ git rebase -i origin/lab<next>
```

References

GDB:

[1] Online documentation: https://sourceware.org/gdb/current/onlinedocs/gdb/
[2] https://www.cs.cmu.edu/~gilpin/tutorial/
[3] https://blogs.oracle.com/ksplice/entry/8_gdb_tricks_you_should

Git:

[1] https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf