Debugging Theory
Lab + Home

Objectives

- Explain general-purpose CPU registers.
- Explain stack.
- Explain debug events.
- Explain soft breakpoints.
- Explain hardware breakpoints.
- Explain memory access faults

Table of Contents

1.

Background Reading

Textbook:

Grey Hat Python, chapters 1, 2

Other resources:
https://docs.python.org/3/library/pdb.html
http://www.gnu.org/software/gdb/documentation/
http://sourceware.org/gdb/current/onlinedocs/gdb.pdf.gz
https://docs.python.org/2/extending/extending.html

Notes common to all lab and home assignment problems

For every lab and home assignment, all work should go into your personal repository, subdirectory named mXX, where XX stands for the module number. For each problem, carefully name the program as described. The programs are extracted from your repository by a Python script, and errors in the program name will result in the instructor never seeing your program, and your mark for it will be ZERO!

There are always many ways how to solve a programming problem, and usually one or two ways which are fast, compact and elegant.

Make sure to push your work to the server often, and have pushed the working version of the program by the deadline specified. The script extracting your programs from your repository will be run at any time after the deadline.

Lab specific intro

In this lab we will explore basic debugging theory.

Problem 1

Download program **`pp_driver.c`**, from the class git repository or from the website. Write C program **`printproc.c`**, which contains single procedure **`printproc()`** ,compile it and create a shared library **`libprintproc.so`**. The procedure will get the parameters as specified in the pp_driver.c, and output the following:

```
State:                      S
ParentPid:                123
ParentGid:                456
StartCode:  0x0000000000400000
EndCode:    0x0000000000067a3c
StartStack: 0x00007fff287e9300
ESP:        0x00007fff287e8818
EIP:        0xffffffff811e74b9
```

Problem 2

Write Python program **m05p02.py** to explore Python to C calling convention, using the Python `ctypes` module. Have the program get the proper parameters about itself (its own process), load the library **libprintproc.so** you just created, and use the **printproc()** to output the parameters.

Problem 3

Write short program in C, **m05p03.c**, which will contain 3 functions with following prototypes:

```
int main(int argc, char **argv[]);

int funtop(int a, int b, char *str);

int funbot(int a, int b, char *str);
```

main() will call funtop(), with parameter values you choose. funtop() will multiply its integer parameters by some constant you choose, and call funbot(). It will simply return the result of funbot(). funbot() will print the str, add the two integer parameters together and return the result.

Each function will print its location in memory. The innermost function will print the contents of the stack, everything from current stack position to the top of the stack. Hint: to find top of the stack, read file /proc/[pid]/maps, look for [stack].

The program gets no parameters from command line.

Problem 4

Place breakpoint just past where you read the current stack pointer, and display the value of registers there.

Clear the breakpoint.

Disassemble the compiled code using `objdump -d` or gdb `disassemble` command, locate the instructions which pass the parameters to function funbot(), and append them to file

m05report.txt

Problem 5

Copy the source code from Problem 3, name it **m05p04.c**, and modify it so that it reliably crashes and generates core dump. Do postmortem analysis using gdb and find out what the locations of stack frames are. Create file **m05report.txt** and cut and paste your debugger output:

```
#0  0x00000000004008b5 in funbottom ...
#1  0x00000000004008f2 in funtop ...
#2  0x0000000000400a8c in main ...
```

Problem 6

Given the C function defined in file `printproc.c`, write Python program **`m05p05.py`** which will take one parameter, a process id, from command line, and call the C function and pass it the proper parameters for selected process.