Lab 6 Exercise 2 - Exploring Function Pointers

Name: NOEL LIM XIAN
Student Number: A0211270Y
Email: E0493357@u.nus.edu
Date: 15 AUGUST 2022

1. Uncomment the block of code with “f = NULL;”, test and undo the changes.
    a. Explain what happens
    Segmentation fault.
    b. Elaborate on the reason.
    f is assigned to NULL (0), and 0x00 is not mapped to the current calling process. (We can execute *proc/<pid>/maps* in code to see the absence of 0x00 in address mapping)
2. What is the difference between the function called by f() at the line with comment LINE1 with f() at the line with comment LINE2. Is the result the same? Explain the similarities and differences between f() and addnum().
    a. Explain what happens
    The result (primitive integer value) is the same.

    b. Elaborate on the reason.
    We can debug using the following *printf* after LINE1 and LINE2 respectively to see that the value of *f* is different after each assignment.

    printf("is addnum == f : %i\n", f == addnum);

    After LINE1, *f == addnum* and after LINE2, *f != addnum*. In fact after LINE2, *f == code_buf*.

    The behavior is the same as the function definition stored in *addnum* is copied into *code_buf* using *memcpy*() before *f* is assigned to *code_buf* and invocation of *f*.
3. What happens when the mmap lines are swapped and uncommented appropriately? Explain the reason for the difference. After testing, the changes should be undone.
    a. Explain what happens
    Segmentation fault.
    b. Elaborate on the reason.
    PROT_EXEC flag is omitted from *memcpy*(). After swapping the *mmap* lines, executing *proc/<pid>/maps* we can observe the output which shows permission not set for execution for the region of *code_buf*. Without *PROT_EXEC* set for the region, code residing in the region is not executable.

```
/* printf("code_buf %p\n", code_buf); */
code_buf 0x7f3a3cb64000
/* system(“cat proc/<pid>/maps”); // pseudo-code */
Memory Map from /proc/213658/maps
/* … omitted impertinent output …*/
```

/* The line below shows perms only “rw-p” for the memory region of the address *code_buf* points to.*/

```
7f3a3cb64000-7f3a3cb65000 rw-p 00000000 00:00 0
```

4. What happened to f() at the line with comment LINE3? Explain the result.
    a. Explain what happens

```
3: f(10)=110
```

The constant term *255* in *code_buf* has been changed to *100*. The consequential code at the point of LINE3 is:

```
/* PSEUDO-CODE */
int code_buf (int a) { // f = (int (*)(int))code_buf;
    return a + 100;
}
```

b. Elaborate on the reason.

Recall contents of *addnum* has been copied to *code_buf* by *memcpy*. *index* searches for the first character-aligned occurrence of *0xff* in *code_buf* which is *255* in decimal. The address of the character occurrence is returned and assigned to *p*. *p* incidentally corresponds to the constant term *255* in the function body. Then, *p* is dereferenced and its value reassigned to *100*.

5. What happened to f() at the line with comment LINE4? Carefully explain the result.
    a. Explain what happens

f(10) will return address of *code_buf*.

Output after LINE4:

```
    printf("code_buf %p\n", code_buf); // code_buf 0x100516000
    printf("4: f(10)=%d\n\n", a); //4: f(10)=77422592
    printf("4: f(10)=%p\n\n", a); //4: f(10)=0x516000

```

f. Elaborate on the reason.

**((char *) code_buf) = 0xc3;* changes the first 16-bit of *code_buf* to the RET instruction *0xc3* in the frame, and the machine interpretes the first 64-bit instruction as RET, resulting in premature termination of *code_buf*. The operation *a+100* is not executed and the return value register remains as the frame pointer (address of *code_buf*).

6. What does the memcpy do? What happens when f() (at LINE5) is called? Is this error different
    a. Explain what happens

Illegal instruction (core dumped)

b. Elaborate on the reason.

instruction copies 2 word-size (bytes) worth of data into *code_buf*. In this case, *data* is {0x0f, 0x0b}.

```
    memcpy(code_buf, data, 2);
```

When function at *f*, which is the same function *code_buf* points to (*code_buff == f)*, is executed the machine interpretes *data* as UD2 instruction as defined in x86 ISA. UD2 generates an invalid opcode description (*https://www.felixcloutier.com/x86/ud*) and the program ends abruptly.

END