Homework0 Written Part

GDB Basics

Q1.1 What memory address does argv store?

```
(gdb) run
Starting program: /home/cc/cs162/sp23/class/cs162-aev/code/personal/hw-intro/map

Breakpoint 1, main (argc=1, argv=0x7fffffffe278) at map.c:16
16        volatile int i = 0;
```

Q1.2 Describe what's located at that memory address. (What does argv point to?)

The argv points to the arguments that were passed to the main function, indicating the Command-Line Arguments.

Q1.3 What is the memory address of the recur function?

1 Point

(gdb) display recur

1: recur = {int (int)} 0x5555555546cd

Q1.4 What values are in all the registers?

```
(gdb) i registers
rax            0x2      2
rbx            0x0      0
rcx            0x0      0
rdx            0x0      0
rsi            0x555555756340   93824994337600
rdi            0x2      2
rbp            0x7fffffffe150   0x7fffffffe150
rsp            0x7fffffffe140   0x7fffffffe140
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x246    582
r12            0x555555554580   93824992232832
r13            0x7fffffffe270   140737488347760
r14            0x0      0
r15            0x0      0
rip            0x555555554701   0x555555554701 <recur+52>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

Q1.5 Which instructions correspond to the return 0 in C?

mov $0x0,%eax

From Source Code to Executable

Q2.1 Generate recurse.S and find which instruction(s) corresponds to the recursive call of recur(i - 1)

```
movl    -4(%rbp), %eax
subl    $1, %eax
movl    %eax, %edi
call    recur
```

Q2.2 What do the .text and .data sections contain?

The .text section contains the executable instructions. And the .data section contains the declarations of initialized data and constants.

Q2.3 What command do we use to view the symbols in an ELF file? (Hint: We can use objdump again, look at “man objdump” to find the right flag).

`objdump -t map.o`

Q2.4 What do the g, O, F, and *UND* flags mean?

g means that it is global. O means that it is the name of an object. F means that it is the name of a function. *UND* means the section is referenced in the file being dumped, but not defined in this file.

Q2.5 Where else can we find a symbol for recur? Which file is this in? Copy and paste the relevant portion of the symbol table.

```
In recurse.o file.

ashby [525] ~/code/personal/hw-intro # objdump -t recurse.o

recurse.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 recurse.c
...
0000000000000000 g     F .text  0000000000000042 recur
...
0000000000000000         *UND*  0000000000000000 printf
```

Q2.6 Examine the symbol table of the entire map program now. What has changed?

Its size is longer than the sum of the symbol tables of map.o and recurse.o.

Q2.7 What segment(s)/section(s) contains recur (the function)? (The address of recur in objdump will not be exactly the same as what you saw in gdb. An optional stretch exercise is to think about why. See the Wikipedia article on relocation for a hint.)

The .text section.

Q2.8 What segment(s)/section(s) contains global variables? Hint: look for the variables foo and stuff.

The .bss and .data sections.

Q2.9 Do you see the stack segment anywhere? What about the heap? Explain.

Both are no. Because the dynamic variables are created from initialization, not written in the segment. What's more, the stack and heap segments do not exist in ELF actually, the main parts of an ELF file are .bss, .text, .data, and .rodata, not including the stack or heap segment.

Q2.10 Based on the output of map, in which direction does the stack grow? Explain.

From high address to low address.
The reason is, from the output of the execution of map, we can see that the i will be execute last, but it has a lowest address, so the growing direction of stack is from high to low.

Starting program: /home/cc/cs162/sp23/class/cs162-aev/code/personal/hw-intro/map
i is 3. Address of i is 0x7fffffffe14c
i is 2. Address of i is 0x7fffffffe12c
i is 1. Address of i is 0x7fffffffe10c
i is 0. Address of i is 0x7fffffffe0ec