Ronan Donovan
Dr. White
Life Beyond Python
10 Nov 2022

Assignment 8

Part I (Look at the Segments in an Executable) - pg 123

1. *Compile the "hello world" program, run ls -l on the executable to get its overall size, and run size to get the sizes of the segments within it.*
    a. Overall size: 16696
    b. Size of segments:

text	data	bss	dec	hex
1569	600	8	2177	881

2. *Add the declaration of a global array of 1000 ints, recompile, and repeat the measurements. Notice the differences.*
    a. *Overall size: 16728*
    b. *Size of segments:*

text	data	bss	dec	hex
1569	600	4032	6201	1839

3. *Now add an initial value in the declaration of the array (remember, C doesn't force you to provide a value for every element of an array in an initializer). This will move the array from the BSS segment to the data segment. Repeat the measurements. Notice the differences.*
    a. *Overall size: 20744*
    b. *Size of segments:*

text	data	bss	dec	hex
1569	4616	8	6193	1831

4. *Now add the declaration of a big array local to a function. Declare a second big local array with an initializer. Repeat the measurements. Is data defined locally inside a function stored in the executable? Does it make any difference if it's initialized or not?*
    a. *Overall size: 20832*
    b. *Size of segments:*

text	data	bss	dec	hex
1814	4624	8	6446	192e

    c. *Is data defined locally inside a function stored in the executable?*
    No it is not stored in the executable as is evident in the data above. The data and bss sections are nearly the same as prior to adding the two large local arrays.
    d. *Does it make any difference if it's initialized or not?*
    It does not.

5. *What changes occur to file and segment sizes if you compile for debugging (-g)? For maximum optimization (-O)?*
    a. *Debugging overall size: 23504*
    b. *Size of sebugging segments:*

text	data	bss	dec	hex
1814	4624	8	6446	192e

    c. *Optimization overall size: 20792*
    d. *Size of optimization segments:*

text	data	bss	dec	hex
1643	4616	8	6267	187b

    - The text segment is most affected by optimization. Compiling regularly and for debugging does not change the segments, only changes the overall size of the file.

Part II (Stack Hack) - pg 127

A. *Compile and run the small test program in a file called stack_hack_1.c with an executable called stack_hack_1.out*

Output: The stack top is near 0x7ffdb7a5cd74

B. *Discover the segment locations in a file called stack_hack_2.c with an executable called stack_hack_2.out*

Static variables are stored in the data segment, and the heap is where dynamic memory allocation takes place.

Output:
The data & text segment are near 0x559873a9e014
The heap is near 0x55987599d2b0

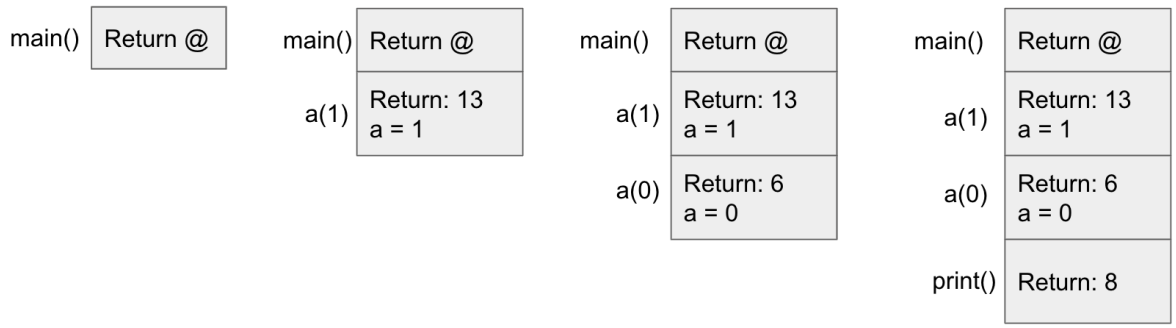
C. *Make the stack grow in a file called stack_hack_3.c with an executable called stack_hack_3.out*

Calling the grow_stack function will declare large arrays and cause the stack top to grow (towards a lower memory address).

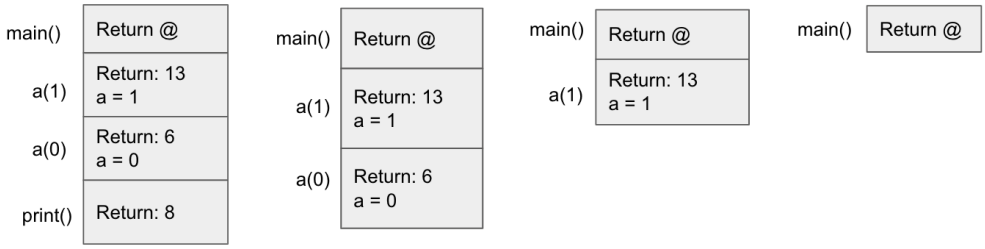
Output:
The top of stack is near: 0x7ffcc9963d14
Growing stack, now top is near: 0x7ffcc98a07ec

Part III (The Stack Frame) - p131

1. *Manually trace the flow of control in the above program, and fill in the stack frames at each call statement. For each return address, use the line number to which it will go back.*



After printing that i reached 0, the stack must be cleared for the program to end



2. *Compile the program for real, and run it under the debugger. Look at the additions to the stack when a function has been called. (my comments are in blue)*

Start of program:
#0 main () at main.c:11

Calling a(1)
#0 a (i=21845) at main.c:3
#1 0x000055555555518f in main () at main.c:12

Calling a(0)
#0 a (i=21845) at main.c:3
#1 0x000055555555516c in a (i=0) at main.c:5
#2 0x000055555555518f in main () at main.c:12

Calling print()

#0 __GI__IO_puts (str=0x555555556004 "i has reached zero ") at ioputs.c:33
#1 0x000055555555517a in a (i=0) at main.c:7
#2 0x000055555555516c in a (i=0) at main.c:5
#3 0x000055555555518f in main () at main.c:12

After print (popping print from stack)
#0 a (i=0) at main.c:8
#1 0x000055555555516c in a (i=0) at main.c:5
#2 0x000055555555518f in main () at main.c:12

After a(0) (popping a(0) from stack)
#0 a (i=0) at main.c:8
#1 0x000055555555518f in main () at main.c:12

After a(1) (popping a(1) from stack)
#0 main () at main.c:13