David Montenegro

Assignment

1. Make a new directory called ASSIGNMENT_08 in your life repo
   mkdir ASSIGNMENT_08
2. Record your answers to each part in a pdf: life/ASSIGNMENT_08/AS08.pdf
3. Push everything to your upstream (GitHub) repo upon completion

Part I (Look at the Segments in an Executable)

1. Make a new directory called Part_I in your ASSIGNMENT_08 that will contain your source code and executables for Part I
   mkdir Part_I
2. Complete *Look at the Segments in an Executable* (Expert C Programming p. 142)
   a. Implement 1 in a file called 1.c with an executable called 1.out
      vi 1.c
      gcc 1.c -o 1.out

   b. Implement 2 in a file called 2.c with an executable called 2.out
      vi 2.c
      gcc 2.c -o 2.out

   c. Implement 3 in a file called 3.c with an executable called 3.out
      vi 3.c
      gcc 3.c -o 3.out

   d. Implement 4 in a file called 4.c with an executable called 4.out
      vi 4.c
      gcc 4.c -o 4.out

   e. Implement 5 in a file called 5.c with an executable called 5d.out (for the debugging question)...
      vi 5.c
      gcc 5.c -g -o 5d.out

   f. ...and an executable called 5o.out (for the optimization question)
      gcc 5.c -O3 -o 5o.out

   g. Record your answers to 1-5 in your AS08.pdf
      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.
         ls -l 1.out
         16696
         size 1.out

text: 1566, data: 600, and BSS: 8

2. Add the declaration of a global array of 1000 ints, recompile, and repeat the measurements. Notice the differences.
   ls -l 2.out
   16720
   size 2.out
   text: 1566, data: 600, and BSS: 4032
   The BSS segment is larger than it was previously.

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.
   ls -l 3.out
   20736
   size 3.out
   text: 1566, data: 4616, and BSS: 8
   The data segment is larger than it was previously.

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?
   ls -l 4.out
   16776
   size 4.out
   text: 1811, data: 608, and BSS: 8
   Data defined locally inside a function is not stored in the executable.
   It does not make any difference if it is initialized or not.

5. What changes occur to file and segment sizes if you compile for debugging? For maximum optimization?
   ls -l 5d.out
   19184
   size 5d.out
   text: 1566, data: 600, and BSS: 8
   For debugging, only the file size changes.
   ls -l 5o.out
   16696
   size 5o.out
   text: 1558, data: 600, and BSS: 8
   For maximum optimization, only the text segment changes.

Part II (Stack Hack)

1. Make a new directory called Part_II in your ASSIGNMENT_08 that will contain your source code and executables for Part II
   mkdir Part_II
2. Complete Stack Hack (Expert C Programming p. 146)
   a. Compile and run the small test program in a file called stack_hack_1.c with an executable called stack_hack_1.out
      vi stack_hack_1.c
      gcc stack_hack_1.c -o stack_hack_1.out
      ./stack_hack_1.out
      The stack top is near 0x7ffeb26fe104

   b. Discover the segment locations in a file called stack_hack_2.c with an executable called stack_hack_2.out
      vi stack_hack_2.c
      gcc stack_hack_2.c -o stack_hack_2.out
      ./stack_hack_2.out
      The stack top is near 0x7ffe4f690024
      The heap top is near 0x5618833ad6b0
      The BSS segment top is near 0x56188322d018
      The data segment top is near 0x56188322d010
      The text segment top is near 0x56188322b081

   c. Make the stack grow in a file called stack_hack_3.c with an executable called stack_hack_3.out
      vi stack_hack_3.c
      gcc stack_hack_3.c -o stack_hack_3.out
      ./stack_hack_3.out
      The stack top is near 0x7ffedaacfff4
      (After growing) The stack top is near 0x7ffedaacf02c

   d. Record all your answers in your AS08.pdf

Part III (The Stack Frame)

1. Make a new directory called Part_III in your ASSIGNMENT_08 that will contain your source code and executables for Part III
   mkdir Part_III
2. Complete The Stack Frame (Expert C Programming p. 151)
   a. Implement 2 in a file called main.c with an executable called a.out
      vi main.c
      gcc main.c -g -o a.out

b. Record your answers to 1-2 in your AS08.pdf

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.
   Stack frame for a called by main:
      Local variables:
      Arguments:
      Ptr to previous frame: Top of the stack
      Return address: Line 10
   Stack frame for a called by a:
      Local variables:
      Arguments: i = 1
      Ptr to previous frame: Stack frame for a called by main
      Return address: Line 3
   Stack frame for printf called by a:
      Local variables:
      Arguments: i = 0
      Ptr to previous frame: Stack frame for a called by a
      Return address: Line 5
   Stack frame for … called by printf:
      Local variables:
      Arguments: “"i has reached zero "
      Ptr to previous frame: Stack frame for printf called by a
      Return address: …

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. Compare with your desk checked results to work out exactly what a stack frame looks like on your system.
   frame at 0x7fffffffe8c0:
   Arglist at 0x7fffffffe8a8, args:
   Locals at 0x7fffffffe8a8, Previous frame's sp is 0x7fffffffe8c0

   frame at 0x7fffffffe8b0:
   called by frame at 0x7fffffffe8c0
   Arglist at 0x7fffffffe888, args: i=1
   Locals at 0x7fffffffe888, Previous frame's sp is 0x7fffffffe8b0

   frame at 0x7fffffffe890:
   called by frame at 0x7fffffffe8b0
   Arglist at 0x7fffffffe868, args: i=0
   Locals at 0x7fffffffe868, Previous frame's sp is 0x7fffffffe890