
   (a) What is the effect of the option `-Wall`?
   (b) What is the effect of the option `-pedantic`?
   (c) What is the difference in the size of the code compiled with and without `-g`?
   (d) What is the difference in the size of the code compiled with `-O1`, `-O2`, and `-O3`?

4. Open the previous C program in the debugger gdb. Set a breakpoint at the first line. Run the program to that point.

   • What is the current ESP?
   • What is the current EBP?
   • What is the current EIP?
   • Use the `x` command to show the assembly language code for the next few commands to be executed.

5. Repeat the previous question, but use the graphical debugger. Show the relevant screen shots.

6. Use the `strace` command to list all of the system calls that your program makes.

7. Use objdump to find the address of the following sections in your program:

   • `.text`
   • `.bss`
   • `.data`

8. Continuing the previous question- create an accurate memory map for your application. In particular, using any appropriate method, locate the address of

   • The environment strings
   • The argv strings
   • `argv`
   • `argc`
   • The start of the stack
   • The bottom of the heap
   • The BSS section
   • The Data section
   • The Code section

9. Write the “Hello World” program described in class. Compile it and link it.

   (a) Include the source, object, and executable code.
   (b) Debug the program. What memory addresses contain the program’s environment strings?
   (c) Debug the program. What memory address contains the program’s arguments?
   (d) Debug the program, and stop it immediately before the syscall for write.
• What is the current ESP?
• What is the current EBP?
• What is the current EIP?

(e) Debug the program. Where is the message string located?
(f) What is the return value for the write syscall? Where is it located?

10. Consider the following two programs:

```assembly
.section .bss
    .lcomm buffer, 32000

.section .text
.globl _start
_start:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

and

```assembly
.section .data
buffer:
    .fill 32000

.section .text
.globl _start
_start:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

Compile each, and compare the size of each program. Explain any differences you see, or explain why there should not be any difference.

11. Consider the program `indexed.s` presented in class. Modify only the text section so that only every second prime number is moved into `eax`.

12. Write a program that uses `execve` to print out the current date and time, using the command `/bin/bash -c date`

13. Write a program that contains the numbers 7, 4, 11, and 16 in the `.data` section. The code should use `push` commands to put this data on the stack; it should then use `pop` commands to put these numbers in `eax`, `ebx`, `cx` and `dl` respectively.

Run your program in a debugger. Break it after the last number is put on the stack

• What is the value of esp?
• Where on the stack are the values stored?

Continue the program, and break it again after all of the values have been pushed to the registers.

• Is the data still on the stack? Explain.
• What is the value of esp?
• What are the values of `ecx` and `edx`. Explain.