

CS220 - Computer System II
Lab 1

Due: 08/31/2017, 11:59pm

1 Introduction

In this lab, you will do the following:

1. Measure the performance of each of for, while and do-while loop.
2. Measure the size of each of the functions (number of bytes).
3. Examine the shared libraries the loop program depends on.
4. Debug a program.

2 Getting Started

In this repository you will find loop.c and buggy.c. You will modify both loop.c and buggy.c.

3 Performance of loops

This task refers to contents of loop.c. Modify loop.c to invoke each of the loop functions separately from main. Create a text file called "Lab1.txt" and report the following:

1. Number of iterations of each of for, while and do-while loops per second.
2. Average time of 3 trials where $n = 1000000000$ for each of the 3 loops.
3. Number of instructions in each of the 3 functions.
4. 2-3 sentences explaining why you think one type of loop runs faster than the other.

Add the necessary commands to the `loop` target in the Makefile in order to build the executable. Refer to Lab 0 if necessary.

Number of instructions in the loop

1. Disassemble the program using:

```
1 $ objdump -d loop > loop.disas
```

2. Open loop.disas using a text editor, search each of the functions and count the number of bytes within the program. You could either take a difference between end address and start address, or you could count the number of instructions manually. Add these counts to Lab1.txt Read the man page for objdump to see other options.

4 Examining the dependencies of loop program

The “ldd” program is used to list all the dlls that a program depends on. Copy the output of the below command to Lab1.txt.

```
1 $ ldd ./loop
```

Add Lab1.txt to the repository and commit with any changes you made to loop.c and Makefile to support your experiments, then push it.

5 Debugging the buggy program

The file buggy.c contains 5 distinct bugs. You are to identify and fix the bugs *without* modifying the intended behavior of the program. The comments should help clarify the intent of the code. Add the following command to your Makefile under the **buggy** target, replacing the error message that is currently present:

```
1 gcc -std=c89 -fno-stack-protector -w -g -o buggy buggy.c
```

Expected output when all 5 bugs are fixed is as follows:

```
1 $ ./buggy
Value of str is c
3 Enter string: The quick brown fox jumped over the lazy dog!
```

```
1 You entered: The quick brown fox jumped over the lazy dog!  
5 New value of x is -6  
x is NOT between 5 and 10  
7 x and str are equal  
All Done!!
```

Running the program in gdb We will be using “gdb”, the GNU debugger to debug the crashes. Load the program using gdb as follows:

```
$ gdb ./buggy
```

Below, you are provided with some common gdb commands. A detailed list of commands is available here: <http://web.cecs.pdx.edu/~jrb/cs201/lectures/handouts/gdbcomm.txt>

Breakpoints are used to pause execution when it reaches a point of interest. Breakpoints can be applied through a function name, line number in a file, or instruction address.

```
1 (gdb) b main
```

This will stop execution when main function begins. Step through statements using gdb command “next”.

```
1 (gdb) next
```

As you step through the code, you can examine the runtime values of different variables using gdb’s print command. For example, in the buggy.c program, typing the command (gdb) p/x str at any point will print the hex value of str at that given point.

Execution occurs on a stack. As control moves from one function to another, the return addresses are saved on a stack. Corruption of the return address is a common source for segmentation faults. When you encounter a segmentation fault, you can examine the execution stack using the following gdb command:

```
1 (gdb) backtrace
```

If you have compiled your code using the ‘-g’ option, you can examine the source code using the command `(gdb) list`. Also, you can examine the assembly instructions by switching to the assembly layout using `(gdb) layout asm`.

Once you’ve fixed `buggy.c`, commit it to your repository.

6 Submitting the result

As with Lab0, after you’ve made your final commit, copy the SHA hash and make a submission on MyCourses with the hash as the comments to the submission. If you are still in class, get the attention of the Teaching Assistant to be graded. If you cannot finish the lab by the end of the lab period, you have until end of day to complete the work in your repository and make a submission on MyCourses.