CS 361: Computer Systems **Lab 2 Worksheet**
Fall 2019

## gdb

1. Before we begin, log in to systems1.cs.uic.edu, and check out the github repository which contains the practice code for this lab session (https://github.com/uicsystems/LabSession2-PracticeCode.git). For this practice session, we are giving you the source code for the files you will be practicing on - in the actual lab assignment, you will get only the binaries.
2. Run make to compile the code.
3. Start running gdb on the "trygdb" binary in gdb with the command "gdb ./trygdb".
4. Here are some useful gdb commands for you to use:
   (a) "run" - runs the attached binary
   (b) "list" - prints out the c source code, with line numbers
   (c) "break linenumber" - pauses the code before executing line linenumber
   (d) "continue" - starts the code again after a breakpoint
   (e) "print variable" - prints the current value of variable. You can also use "p variable".
   (f) "watch variable" - will break when the variable you're watching changes
   (g) "set variable=value" - sets a variable to the value you specify
5. Use the above commands to answer the questions about gdb on gradescope.

## strace

1. strace is a debugging program that tells you all of the **system calls** a program makes. System calls are requests a program makes to the operating system - we will go over them in depth in class this week.
2. Run strace on the trystrace executable with "strace ./trystrace".
3. strace will print out a bunch of systems calls, most of which are actually the operating system creating, loading and running the executable. The first line of output we're interested in looks like "write(1, "Hello, operating system", 24Hello, operating system
   ) = 24"
   Note that this is actually one line ("write(1, "Hello, operating system", 24) = 24") being interrupted by our program printing.
4. This line tells us that the system call "write" was called (as a result of our printf call) on the parameters "1, "operating system", 24", and had the return value of 24.
5. Some helpful strace options to play around with:

   (a) “strace -e write” will only show you the calls to write. (You can substitute your favorite system call for write.)
   (b) “strace -f” will trace any processes spawned by the original process.
   (c) “strace -p pid” will attach strace to the process ID of a running process.
   (d) “strace -s x” will print the first x characters of any string (useful because strace truncates strings by default.)

6. Answer the questions about strace on gradescope.

## ltrace

1. ltrace is much the same as strace, except that it prints out calls to shared libraries, rather than system calls.
2. Run ltrace on the tryltrace executable with “ltrace ./tryltrace”.
3. Notice that the ltrace output is almost identical to strace, except that it prints out library calls, rather than system calls. Try running both strace and ltrace on both binaries to see the differences in what they print out.
4. All of the strace options listed above will also work for ltrace.
5. Answer the questions about ltrace on gradescope.
6. You should now be able to find the secret codes for binaries 0-3 in lab 2. We will go over what you need for binary 4 next week.