*1. Signals Lab*

**Instructions**

Your job is to implement a simple program that will register signal handler functions, then test signal handlers by issuing a signal from the command line (i.e. Ctrl^C) SIGINT signal.

Use the man pages (i.e. $ man signal) to find out all the possible signals our system provides. This will also provide practice with reading manual pages (or man pages for short). These are extremely useful for understanding these system calls at a deeper level. For instance, what arguments do the functions expect? What format should the arguments be? What does this function return?

1. signal.c - Simple C program that contains the source code for the main executable.
2. signal - executable that you should be able to run in your shell.
3. Makefile - Makefile for the main executable. Run make signals. program.

Here you will write a sequence of simple C program that will teach you the basics of signal handling.

A signal is a operating system mechanism that enable programs to be “signaled” to take specific actions. You are probably already familiar with some UNIX signals; for example, by pressing CTRL-C on the terminal, you are directing the operating system (OS) to deliver a SIGINT signal to the running program, which usually has the effect of terminating the program. Similarly, if you press CTRL-Z, you are directing the OS to send a SIGTSTP signal, which usually has the effect of stopping a program so it can be resumed later.

In this lab, we will investigate a different class of signals whose purpose is to not terminate a program, but rather instruct the program to take an action. Particularly, you will employ the SIGALRM signal to take periodic actions, such as printing a message to the screen. To help you get started, I’ve provide a basic “Hello World” program, signal.c which you will build upon to complete this part of the lab.

There are two key function calls in hello signal.c: signal() and alarm(). The alarm() system call instructs the operating system to deliver a SIGALRM signal after n seconds, n being 1 in this example. The signal() system call instructs the operating system to execute the function handler when the SIGALRM signal is delivered. Now, it is clear that the “Hello World” program sets up a signal handler for SIGALRM, line 15; a timing for the delivery of the SIGALRM, line 16; busy waits for the signal to be delivered, line 17; and once the signal is delivered, the signal handler is invoked, printing “Hello World” and exiting, line 7-11. At this point you should compile and execute hello signal.c and observe the timing of the output — it is delayed by 1 second.

This style of programming with signal is based on the principals of preemptive execution. That is, the execution of the handler function preempts the main execution of the program. Once the signal is delivered (during the busy wait), program execution jumps to the handler function, and once the handler returns, execution jumps back to the point where the main execution was preempted (or would have, if there was not an exit() call). This is a very powerful (and often confusing) programming paradigm, which you will use throughout this lab.

**Signal Handling Programming Problems**

Program solutions to the following problems by extending signal.c:

1. Change signal.c such that after the handler is invoked, an additional printf("Turing was right!\n") occurs in main() before exiting. You will probably need to use a global variable and change the condition on the while loop.
2. Change signal.c such that every second, first “Hello World!” prints from the signal handler followed by “Turing was right!” in main(), over and over again indefinitely. The output should look like:

Hello World!

Turing was right!

Hello World!

Turing was Right!

...

1. Program a new program timer.c that after exiting (via CTRL-C), will print out the total time the program was executing in seconds. To accomplish this task, you will need to register a second signal handler for the SIGINT signal, the signal that is delivered when CTRL-C is pressed. Conceptually, your program will request a SIGALRM signal to occur every second, tracking the number of alarms delivered, and when the program exits via CTRL-C, it will print how many alarms occurred, and the number of seconds it was executed.

When complete push your code to github, and post a link to your github in Canvas.