CSCI 4969/6966 Assignment 2

Simple Message Server/Client

Due Monday October 2nd, 11:59:59 PM

In this assignment you will be writing a simple server and client using POSIX C sockets, and then exploring the underlying calls. This is an **individual assignment**.

Basic Requirements

Write two files, **server.c** and **client.c** that are the server and client respectively for this assignment. The server should support multiple clients by using *fork()*. The server is invoked by running it with a port number as the argument, and the client is run with a port number and a file to write output to. For example:

```
./server.out 1234
./client.out 1234 log1.txt &
./client.out 1234 log2.txt
```

If the incorrect number of arguments is provided, you should print a proper usage message. If any of the network calls fail, an error message should be printed. The server should send 5 messages over TCP, each spaced apart by 2 seconds. Each message starts with the port being used for the message wrapped in brackets, then "PID" and the process's PID, and then "Iteration #" followed by a number 0-4 showing the message count. An example is:

```
[56757] PID 736 Iteration #0
[56757] PID 736 Iteration #1
[56757] PID 736 Iteration #2
[56757] PID 736 Iteration #3
[56757] PID 736 Iteration #4
```

The client should write anything it receives to the file passed in. The client should not make any assumptions about how long the server is waiting between sending messages, nor should the client assume exactly 5 messages. Your code

should work if either the number of messages or the time between messages is changed in the server code.

Both sides should close any relevant sockets/files when they are done. The server should run indefinitely until a *SIGINT* (CTRL+C) signal is sent, at which point it should exit gracefully. You may need to use *signal.h* to catch the signal so you can do cleanup.

Exploring the Kernel

Once you have written your code and are satisfied with it, you will need to go exploring the kernel source. Try and figure out where the data is sent/received at the application layer, and work your way as far down the network stack as you can. It is very unlikely you will find all the calls in a reasonable amount of time, or that you will understand every line of code you traverse. This is fine - the goal is not to have a perfect understanding, but to get a feel for the code and a basic understanding of what happens on the kernel-side in this seemingly simple program.

Submit a writeup describing how you tested your code, and then explain where the **network-related** code enters the kernel and as much of its path through the kernel from application to physical network device (function calls, variable changes) as you were able to figure out. Rather than giving a list of calls, you should explain this in prose (complete sentences!), mentioning function names, variables, etc. when needed. Describe what these functions are doing, why they're being called, and how you came across them. Make sure to give paths for ambigious filenames/functions! You can refer to lines of code by filename:line, i.e. fork.c:250. Do not explore what *fork()* does.

To get started, you will find **strace** to be very helpful. Simply run your program as normal, but prepend it with **strace**. This will print out all system calls requested during the execution of the program. Don't forget your program's arguments! You may also want to refer back to Lecture 4 to remind yourself how system call declarations look. Another idea is to look for keywords specific to your application, such as PF_INET or SOCK_STREAM.

Submission to Submitty:

1. **server.c** and **client.c** source files
2. **README** - A text file with any information I need to know about running your program, miscellaneous notes, any sources you referenced when writing your code, etc.
3. **writeup.pdf** - A PDF file with your writeup from the "Exploring the Kernel" portion of the assignment. Minimum of 2 pages.