CS4700/CS5700 Network Fundamentals
Project 1: Simple Client

Submission deadline: 11:59pm Wednesday November 28th

0.1 Description

This assignment is intended to familiarize you with writing simple network code. You will implement a client program which communicates with a server using sockets. The server will ask your program to solve hundreds of simple mathematical expressions. If your program successfully solves all of the expressions, then the server will return a *secret flag* that is unique for each student. If you receive the *secret flag*, then you know that your program has run successfully, and you will receive full credit for the assignment.

0.2 Programming Language

You can write your code in whatever language you choose, as long as your code compiles and runs on **unmodified** CCIS Linux machines **on the command line**. Do not use libraries that are not installed by default on the CCIS Linux machines. Similarly, your code must compile and run on the command line. You may use IDEs (e.g. Eclipse) during development, but do not turn in your IDE project without a Makefile. Make sure you code has **no dependencies on your IDE**.

0.3 Protocol (updated)

The server runs on the machine *login.ccs.neu.edu* and listens for requests on a TCP socket bound to port 27993 (non-SSL). *login.ccs.neu.edu* has a publicly routable IP address, so you should not have any problems connecting to it from campus or home.

This exercise has four types of messages: HELLO, STATUS, SOLUTION, and BYE. Each message is an ASCII string consisting of multiple fields separated by spaces (0x20) and terminated with a line feed (0x0A, \n). The maximum length of each message is 256 bytes. Messages are case sensitive.

The protocol works as follows. The client initiates the protocol by creating a TCP socket connection to the server. Once the socket is connected, the client

sends a HELLO message to the server. The format of the HELLO message is:

```
cs5700fall2015 HELLO [your NEU ID]\n
```

In your program you should replace [your NEU ID] with your NEU ID (including any leading zeroes). You must supply your NEU ID so the server can look up the appropriate secret flag for you. The server will reply with a STATUS message. The format of the STATUS message is:

```
cs5700fall2015 STATUS [a number] [a math operator] [another number]\n
```

The three variable fields represent a simple mathematical expression, e.g. "5 + 10". The server may return plus, minus, multiplication, or division expressions. All numbers will be between 1 and 1000. Your program must solve the mathematical expression and return the answer to the server in a SOLUTION message. The SOLUTION message has the following format:

```
cs5700fall2015 [the solution]\n
```

It is okay for the solution to be negative. In the case of division, round the answer down to the nearest integer (do not send floating point numbers to the server). The server will respond to the SOLUTION message with either another STATUS message, or a BYE message. If the server terminates the connection, that means your solution was incorrect. If the server sends another STATUS message, your program must solve the expression and return another SOLUTION message. The server will ask your program to solve hundreds of expressions; the exact number of expressions is chosen at random. Eventually, the server will return a BYE message. The BYE message has the following format:

```
cs5700fall2015 [a 64 byte secret flag] BYE\n
```

Once your program has received the BYE message, it can close the connection to the server. If the server returns "Unknown_Husky_ID" in the BYE message, that means it did not recognize the NEU ID that you supplied in the HELLO message. Otherwise, the 64-byte string is your *secret flag*: write this value down, since you need to turn it in along with your code.

0.4 Your Client Program

Your client program must execute on the command line using the following command.

```
$ ./client <-p port> <-s> [hostname] [NEU ID]
```

Your program must follow this command line syntax **exactly**, i.e. your program must be called *client* and it must accept these exact parameters in exactly this order. If you cannot name your program *client* (i.e. your program is in Java and you can only generate client.class) then you **must** include a script called *client* in your submission that accepts these parameters and then executes your actual program. Keep in mind that all of your submissions will be evaluated by grading scripts; if your program does not conform **exactly** to the specification then the grading scripts may fail, which will result in loss of points.

The *-p* port parameter is optional; it specifies the TCP port that the server is listening on. If this parameter is not supplied on the command line, assume the port is 27993. The *-s* flag is optional; if given, the client should use an SSL encrypted socket connection. Your client only needs to support *-s* if you are trying to get the extra credit points. The [hostname] parameter is required, and specifies the name of the server (either a DNS name or an IP address in dotted notation). The [NEU ID] parameter is required. Your code must support NEU IDs that have leading zeroes (do not strip them!).

Your program should print **exactly one line of output**: the *secret flag* from the server's BYE message. If your program encounters an error, it may print an error message before terminating. **Your program should not write any files to disk**, including writing to the *secret_flags* file.

0.5 Other Considerations

You may test your client code with our server as many times as you like. Your client should conform to the protocol described above, otherwise the server will terminate the connection silently. Your client program must verify the validity of messages by strictly checking their format, i.e. the server may send corrupted messages just to try and crash your software. If a received message is not as expected, such as an incorrect field or wrong message type, you must assert an error and terminate your program. You should be strict; if the returned message does not exactly conform to the specification above, you should assert an error. Remember that network-facing code should be written defensively.

0.6 Submitting your Project (Updated)

To turn-in your project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, any challenges you faced, and an overview of how you tested your code.
- A file called secret_flags. This file should contain full names, NU IDs and the secret flags of all group members, one per line, in plain ASCII, i.e.:

  ```
  name_1  NU_ID_1  secret_flag_1  [SSL_flag_1]
  name_2  NU_ID_2  secret_flag_2  [SSL_flag_2]
  ```

  where SSL_flag_* is the second secret flag for the SSL enabled client (optional).
- Your README, Makefile, secret_flags file, source code, etc. should all be placed in a single zip file. You submit your project by uploading your zip file to Blackboard (see Project 1 folder on Blackboard).

Only one group member needs to submit your project. There is no need to register your group just make sure your README file and secret_flags includes all the group members names and NU IDs. Your group may submit as many times as you wish; only the last submission will be graded, and the time of the last submission will determine whether your assignment is late.

0.7 Grading

This project is worth 5 points. If your program compiles, and you successfully submit the secret flags of all group members, then you will receive full credit. We will randomly check student's code to make sure that it works correctly. All student code will be scanned by plagarism detection software to ensure that students are not copying code from the Internet or each other.

0.8 Extra Credit

It is possible to earn 1 extra credit point on this assignment. To get the extra credit point, you must modify your client such that it supports SSL connections. If the *-s* parameter is given to your program, it should connect to the server

using an encrypted SSL socket and complete the protocol normally (i.e. HELLO, STATUS, SOLUTION, and BYE). You may assume that the server's SSL port is 27994, unless the port is overridden on the command line using the *-p* option.

When you successfully run your SSL-enabled client against the SSL version of the server, you will receive a new secret flag (that is different from the normal secret flag). You and your partner add these SSL secret flags into the *secret_flags* file when you turn in your project. You will only receive the extra credit point if your code successfully implements the *-s* option, and you include the SSL secret flags in the secret_flags file.