COMPSCI 230 Computer Systems Principles

Math Bot

Overview

This assignment will give you the opportunity to write a computer networking client that will connect to a remote server and interact with it. You will be writing this assignment in the C programming language. To summarize, the skills that you will build are:

- **Network System Calls:** You will be using system calls in order to communicate to the network. These are special functions that act as an API to the kernel and the operating system. You should take a look at this post to see a list of network system calls that are used in building networked applications. Note, you will not be using all of these, but rather a subset of them. In addition, along with these system calls you will need to become acquainted with the various structures that are used as parameters and return values for these system calls.

- **Networking:** This assignment will use a network socket, which is a special file descriptor that allows you to interface the networking stack. As explained in class, computer networks are organized in layers. The sockets we will be using here will allow you interact with the transport layer of the stack. More specifically, you will be using TCP stream sockets.

For help with this assignment we recommend that you review the course material as well as read the Unix Socket Tutorial to better understand the system calls and structures used in Unix network programming.

Source Files and Compilation

You are not provided any starter code with this project. Because we are using an autograder to score your work, you must, however, provide all code that you write in a single C source file called **`client.c`**. In fact, this is the only file you need to submit to Gradescope! You do not need to submit a Makefile for this project or a README.txt. The autograder will use **`gcc`** to compile your submitted C file, so make sure it compiles in your vagrant environment before submitting.

Details and Objective

You will implement a network client that will communicate to a remote server in the C programming language. The server implements a simple protocol that requires your client to send your **`netid@umass.edu`** email. After you do this then the server will respond to your client with a series of simple math problems that your client will need to solve (how you solve the math problems is up to you). The steps are outlined as follows:

Your client will be expected to conduct the following procedure:

- Step 1: Open a TCP stream socket
- Step 2: Connect to the remote server on the specified port number
- Step 3: Send your SPIRE ID to the server in the following format: NETID@umass.edu
- Step 4: Receive the first math problem
- Step 5: Send the CORRECT solution to the server (Server will drop connection if it is wrong)
- Step 6: Continue steps 4 and 5 for a random amount of times (No less than 300, but no more than 2000)
- Step 7: Once step 6 is completed, you will receive a 64-byte string (flag) that is unique to your NetID. Once you capture the flag you know you have implemented your client correctly.

Socket Requirements

Your client is expected to be written in C, not a high-level language such as python (no matter how tempting that might be). In order to write a networking program in C, you must use various system calls provided by the socket API. In particular, the system calls you might find useful are:

```
socket(int domain, int type, int protocol);
ssize_t send(int socket, const void * buffer, size_t length, int flags);
ssize_t recv(int socket, void * buffer, size_t length, int flags);
ssize_t connect(int socket, const struct sockaddr *address, socklen_t address_len);
int close(int fildes);
```

If you are unfamiliar with any of these, you should read up on it, review the course material and example code, look at the man pages. Reviewing the man pages will be very helpful during the completion of this assignment. Please do not neglect this valuable resource. Note: These system calls are functions, and just like any function they can possibly return errors. Be sure to check the return values of these functions to make your client more robust.

To make it a little easier on you we provide you with the minimum list of header files that you need to include in your client in order to produce a working client:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
```

Notice that `connect` takes as an argument: `struct sockaddr *address`. This is one of the challenges of the assignment. You will have to populate the struct with the **IP address** and **port number** as well as other information. An extremely helpful link for this is found in our code example with slides. Please read that entire code closely! It is very short, but dense. **Hint:** The network requires big endian format, so you will have to convert between the two. Helpful functions for this are `htons` and `inet_pton`. They do have man pages available to anyone with a Linux (vagrant) or Mac distribution!

Protocol Requirements

To capture the flag from the math bot server you must combine your understanding of the socket API functions and an application-layer protocol known as **math speak** which is as follows:

When you first connect to the server you must identify yourself. In particular, you must send the following string:

```
cs230 HELLO <NETID>@umass.edu\n
```

This string must be **exactly** as we describe above - network protocols are very specific. You must replace `<NETID>` with your UMass **NetID**. By sending the identification string it will initiate the math bot server to start sending you math problems. You will immediately receive a "status" message with the following format:

```
cs230 STATUS NUM OP NUM\n
```

This status message includes a simple arithmetic operation. An example of an actual message is:

```
cs230 STATUS 505 * 700\n
```

You will need to implement functionality in your client that will compute the math problem provided in the server's status message. After you do that you need to send a response back to the math bot server that is formatted like this:

```
cs230 <ANSWER>\n
```

You must replace <ANSWER> with the answer to the math problem. Here is an example:

```
cs230 353500\n
```

With your response back to the server, the server will then repeatedly send you hundreds of math problems that your client must solve. Your client will need to solve each of the status math problems until you receive the response with the flag:

```
cs230 <FLAG> BYE\n
```

The <FLAG> is a long hash value. Here is an example of the final message you will receive from the server before it disconnects from your client:

```
cs230 7c5ee45183d657f5148fd4bbabb6615128ec32699164980be7b8b451fd9ac0c3
```

If you are able to "capture the flag" you have completed this assignment successfully. You still need to submit it to Gradescope though.

Program Requirements

Your client program must accept the following command line arguments in this order:

1. Identification
2. Port
3. Host IP address

The first argument (Identification) must be a UMass email address of the form "NetID@umass.edu". The port and host IP address are as they are defined by the socket API.

We will be running a test server at address 128.119.243.147 on port 27993. You are welcome to test your client by running it like so:

```
$ ./client netid@umass.edu 27993 128.119.243.147
```

And see if you can capture your flag! **Note:** it took 22 seconds for our solution client to complete over a 5 Mbps connection. If it seems to take a while, it may not be wrong. If it never ends you should then add debugging output to see where it is getting blocked.

Hints and Suggestions

Here is a list of hints and suggestions that will help you in completing this assignment:

1. Print the messages that are being sent and returned from the client and server respectively to see what you are sending to the server and what the server is sending back to you.
2. Make sure the messages you send to the server are *exactly* as stated above. Do not add any extra spaces or additional newlines (just the one that is at the end of the message) - any extra bytes will cause a failure in communication.
3. Figure out how to extract the math problems, evaluate them, and send back the correct result. Pay careful attention to division - we expect the result to be truncated. For example, if your client is asked to solve "200 / 3" you must respond with "66" as the result, not "66.66666" or "67".
4. Figure out how you can identify when your client is done solving math problems.
5. Make sure your compiled client accepts the command line arguments exactly as we described above. If it does not, the autograder on Gradescope will give you a 0.
6. Print the final message so you can see the flag and verify that you have implemented the client properly. Do not forget to submit to Gradescope!

Grading Breakdown

1. Gradescope Submission and Autograder (1000 points)

NOTE: There is no video for this last submission!

Submitting Your Solution

This assignment is a bit different than the past assignments. You will need to submit only the `client.c` file mentioned above to Gradescope. The autograder will try to compile your C code and it will then run a local server (the same as the public server) to see if your client passes the tests. The tests are simple: (1) connect to the server, (2) solve the math problems, and (3) capture the flag. If you are able to do this successfully you will successfully complete this assignment. The total number of points for this project is 1300 (it doesn't mean that it has more weight than other projects).