CS 232 Operating Systems

Client-Server Computing

Overview

Most networking applications consist of two components:

- a *server* that resides on a remote host, providing whatever services are required by the application; and
- a *client* that a user of the application uses from their local computer to interact with a remote server.

For example, if you run **ssh** (or **scp** or **sftp**) to connect to a remote host, you are actually invoking a client for that service, which contacts an ssh-server on the remote host. As you type commands at the client, they are relayed to the server, which executes them and relays their results back to the client, which displays those results.

This project is to write the client and server programs for a **Caesar Cipher** service.

Start by Accepting this Assignment. Then, clone your repo.

Details

When your client connects to your server, the first thing the client sends is an integer *n* in the range 1..25. The server replies with this same number to confirm its reception. For any other lines of text the client sends, the server rotates the characters in those lines of text *n* positions and then returns the rotated text. If the client initially sends something other than an integer in the range 1..25, the server should respond with an error message and close the connection.

Note that non-alphabetic characters should not be rotated -- our version of the Caesar Cipher only affects alphabetic characters (a-z and A-Z).

Also, note that the lower-case letters stay lower-case letters and upper-case letters stay upper-case letters -- i.e., you do not rotate a lower-case letter into an upper-case letter, or vice versa.

Note also that if we view this service as a function, one session can serve as the inverse of another session if the first session uses *n* for its rotation amount and the second session uses

26-n for its rotation amount. Your service should thus be able to both encode and decode messages using the Caesar Cipher.

Part A

You are to write a client program that (after displaying an appropriate 'welcome' message), asks for and reads the rotation amount from the user. Using a socket and the TCP protocol, your client should connect to a CaesarCipher server and send it whatever rotation amount the user specified. Your client should receive and display the server's response. Your client should continue reading the user's input, sending it to the server, receiving the server's response, and displaying that response, until the user types *quit*, at which point your client should close its socket and terminate.

Your client should get the server's hostname and port via command-line arguments. To test your client, you may use my CaesarCipher server that is running on *cs232.cs.calvin.edu* at port *9876*; e.g.:

```
$ java CaesarCipherClient cs232.cs.calvin.edu 9876
```

Part B

You are also to write your own CaesarCipher server. Your server should open a TCP socket, and listen for a client's connection. When contacted by a client, your server should spawn off a new thread to handle the session with that particular client, and then listen for the next client connection.

Your thread should receive a message from the client. If the first message is an integer *n* in the range 1..25, your thread should save *n* and send it back to the client; otherwise, it should reply with an error message and close its socket.

If it has processed a correct first message, your thread should continue to receive messages from its client and reply with those messages rotated.

Your server should get the port it is to use as a command-line argument; e.g.:

```
$ java CaesarCipherServer 9876
```

To help trace its execution, your server should display the date, time, and the IP address of the client each time it receives a new connection.

You may write your system in any of the following languages:

- Python: writing networking code in Python is quite easy. Highly recommended.

- Java: provides an easy means of building this system, thanks to its standard Thread, Socket and ServerSocket classes. These classes allow you to easily define sockets as distinct objects and manipulate them via object-oriented methods. There are numerous tutorials on Java network programming online, including:

  https://docs.oracle.com/javase/tutorial/networking/, and

  https://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html

- C++ provides the hardest way to build the system, because there are no standard C++ network interface classes. Instead, the BSD-Unix (C) socket library is used, that allows you to create and manipulate a socket via function calls. C socket programming is thus considerable 'lower level' than Java socket programming. A brief tutorial on C/C++ socket programming is available at:

- http://www.yolinux.com/TUTORIALS/Sockets.html

Tips

1. Begin by writing a client that works correctly with my server.
2. When your client is working correctly, write a simple echo **server** that echoes back to the client whatever it sends.
3. Make your echo server multithreaded, so that it can handle multiple connections simultaneously.
4. Modify your multithreaded echo server so that when the first string it receives is *n*, it rotates the chars in all subsequent strings it receives before sending them back to the client. From there, it should be easy to complete all remaining requirements.

Hints

For the client side, I used these Java classes:

- `Socket` (and `SocketException`)
- `DataOutputStream`
  - `writeBytes()` to write out to the stream.
- `BufferedReader`
  - `readLine()` method: to read from socket and standard input (`System.in`). Note that it removes any newlines at the end of the string, and returns null if the socket was closed.
- `InputStreamReader`

- `IOException` and `UnknownHostException`.

For the server side, I used these Java classes:

- Most of the above, plus...
- `Date`, `DateFormat`, `SimpleDateFormat`
- `ServerSocket`
- I used `(int) 'a'` (e.g.) to convert a character to its corresponding integer, and `(char) anInt` to do the reverse.

Submission

Don't forget to push your changes up to your github.com repo.

Note: This assignment will not be accepted following the due date.

This page is maintained by Victor Norman.