**CS 451**
**Fall 2024**
**Assignment 2**
**Due:** September 23rd at 11:59pm CST/CDT
**Worth 100 points**

**Objective:**
There are a few of objectives to this assignment beyond just a simple socket programming assignment:

- It will help you "tune up" your programming skills and prepare you for other assignment in the course
- It will put practical experience into one of the models we discussed, namely Client Server.

**Part1: Client Server Hangman**
If you don't know what hangman is, you can play it here
http://www.billsgames.com/hangman/

We will be writing a client server version of Hangman in C/C++ or Java.

Your game is to be non-graphic, so you don't need to draw a hangman; this is how games where when we had dial-up. ☺

Both programs will be text based. The basic program logic is as follows:

When the server starts up, it loads all the words in file in Canvas. There's 69k words or so. Your program needs to store all the words because you can play N rounds.

A server creates a Socket listener on a certain port (you decide which port), and the client connects to that port (you pass in the server and port to the client program). When a client connects to a server, a new thread is created on the server side to support each new client.

When a new thread is created to run a new client game, a word is chosen from the words collection at "random".

At the start of each client session, two strings are sent back to the client, such as (assume the word is "assume"):

"- - - - - -"
"You have 10 lives left."

The client then prompts the user to enter a letter. Say the user types an "s". The "s" is sent back to the server thread, and the server looks to see if there is a hit in the chosen word

"assume" (the search is CASE INSENSITIVE). The letter "s" is added to the list of already-asked letters on the server. There is an "s", of course, so the strings sent back to the client would be:

"- s s - - -"
"You have 10 lives left."

Now, the client displays this information and prompts the user to enter another letter. Suppose the user types a "z". The "z" is sent back to the server thread, and the server thread checks the word for a match. There is no z in the word "assume", so the server thread decrements its lives counter from 10 to 9, and adds the letter "z" to the list of already-asked letters. So the server would then send back the following strings:

"- s s - - -"
"You have 9 lives left."

The client and server continue to communicate until either the word is correctly guessed, in which case the server thread will respond:

"Congratulations! The correct word was indeed 'assume'"
"Want to play again?"

Or, in the case of failure, the server would respond:

"Sorry, you have used up all 10 of your lives. The correct word was 'assume'".
"Want to play again?"

The client will display these strings, and will close the socket connection. If the user wants to play again, a game begins anew. If the user is done playing, the client exits gracefully.

- The server will also need to be able to handle client disconnects and timeouts since the Internet is the Internet.

- The server needs to give an error and continue if more the 1 character is entered, or if non-alpha is entered.

- Please use TCP not UDP

- Feel free to write locally and use localhost to test.

**Part2:**
Deploy 2 AWS or https://www.chameleoncloud.org vm's.
Compile and deploy your code.
Execute your code and show a few games as well as a graceful, and ungraceful exit.

Take screen shots of each of the steps above.

**Write up:**

1. Explain how this fits into the client server architecture
2. How would this work differently without threads? How would it work differently going from Linux to Windows, if there are differences?
3. Explain the flow of your implementation and what design choices you made
4. I asked for TCP – what is the difference between tcp and udp and what additional logic would you need for UDP.

**What to submit:**

**A zip file with:**

**Within the zip file:**

The server file name will be IIT_USERNAME-server.{c,cpp,java} (ie. blenard_server.c)
The client file name will be IIT_USERNAME-client.{c,cpp,java} (ie. blenard_client.c)
**Please include your name Email Address in each program file you submit (as a comment).**

**Screen shots of the program being compiled, executing one two separate systems (Part 2), and terminating gracefully.**

The writeup as a txt, PDF or DOC/DOCX.

**Grading Guidelines**

In addition to correctness, part of the points count towards how well code is written and documented.

**33 pts**: client
**33 pts**: server
**34 pts**: Write up and documentation in the code. This also includes screenshots (no screen shots, -5 per missing one).