Wissam Razouki

CSE 3300

Programming Assignment #1: Jumble

**Description:**

In this assignment, we are tasked with creating and implementing a server that supports multiple clients playing a game called Jumble. The server’s task is to give clients that are connected to it a random sequence of letters and wait for a response. If the client gives the correct unscrambled word, they win, and must solve the next Jumble. If not, they are given the correct answer and another Jumble. While this is occurring, the server should also be waiting for additional clients. Clients connecting to the server should not interfere with each other in any way.

To accomplish this, I used the provided server, client and Jumble codes. The server and client start by setting up the appropriate TCP connections. The current date and time are printed on both ends. The server then opens the wordlist.txt file and converts it into a list of word strings. When a client connects, a new thread is made and the client’s information is printed out on the server side. The server picks a random word from the word list, scrambles the letters, sends it to the client, and waits for an answer. The client receives the jumbled word and it is printed out on the client side. Once an input is provided, the program checks if it’s an empty string. If so, the socket is closed and the connection to the server ends. If not, the client sends the input to the server. When the server receives the answer, it checks to see if it’s a valid word (by checking if it’s in the word list) and if it matches the original unjumbled word. If so, the server sends ‘You win.’. If not, the server sends the correct word. The process then repeats until an empty string is given or the client is closed. While this is occurring, the server is always waiting for new clients to connect. When there’s a new connection, the server starts a new thread to handle the new client.

**Tradeoffs:**

Since this was a simple program, there weren’t many tradeoffs that I considered or made. I’m certain there are better ways of implementing the Jumble server code such that it can give clients a faster response. For example, instead of using the “+” operator to concatenate strings, I could use a faster method. However, to maintain the simplicity of this program, I decided to keep most of the code as is.

**Extensions:**

There are many ways of extending/improving this program. For example, instead of having only single-player games, we can code the server to keep track of a certain number of clients competing against each other. To accomplish this, we can design the server to first wait for a certain number of clients to connect before initiating the Jumble game. Once enough players have joined, the game starts. If more clients want to connect, a separate game must be made. While players are solving the words, the server keeps track of each players score based on how many correct answers it receives from that player. When the maximum score is reached, the corresponding player wins.

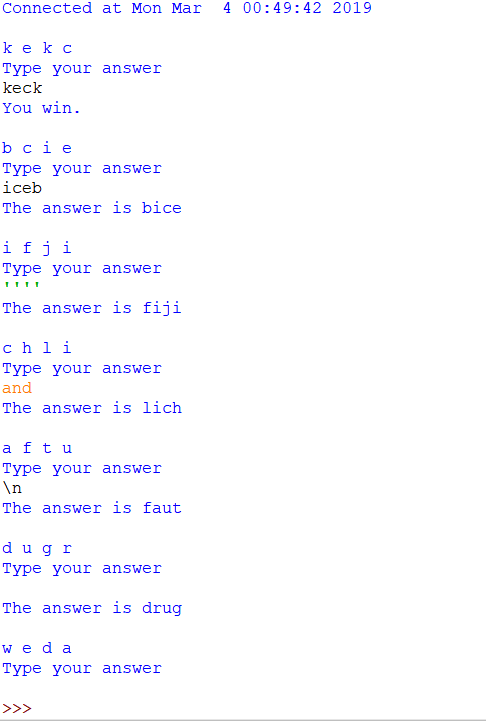
We can also have the client select a preferred difficulty based on the number of letters in each scrambled word.

**Test cases:**

Test case #1: Single-client connection

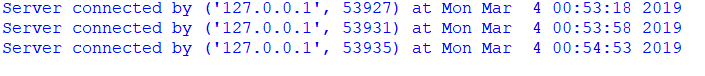
This is a very simple test case that checks to see whether the program is working properly or not.

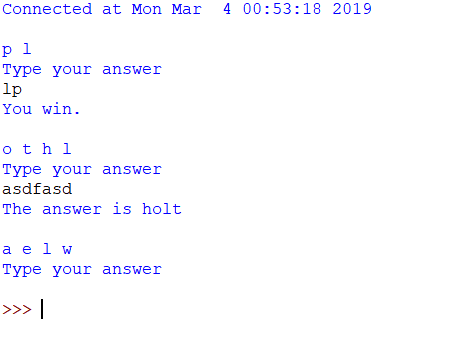
1. One client connects to the server. The server should send Jumbled words to the client. If the client provides the correct answer, the server sends ‘You win.’ and provides the next Jumble. Otherwise, the server sends the correct answer and another Jumble. If the client gives an empty response (by pressing the Enter key – whitespaces don’t count), the connection ends.
2. I chose this test case to see if the server works at the most basic level.
3. The outputs are as follows:
   1. Server: 
   2. Client:



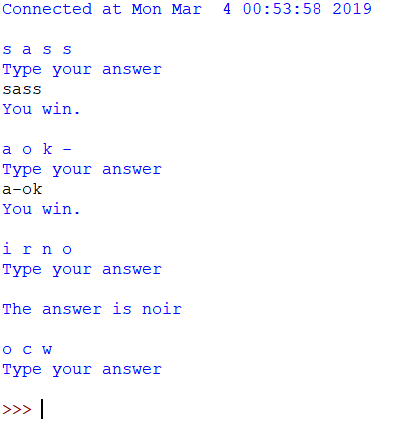
Test case #2: 3-client consecutive connection

This test case checks to see whether the server can handle consecutive connections.

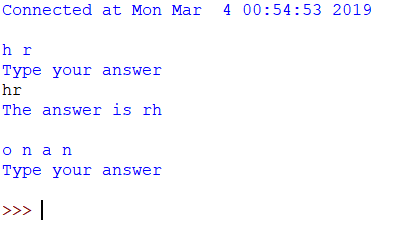
1. First, I will connect a client to the server, solve a few Jumbles, then press Enter to end the connection. Then, I will do the same for the next two clients.
2. I chose this test case to see if the server works with multiple consecutive clients with no errors.
3. The outputs are as follows:
   1. Server: 
   2. Client 1:



* 1. Client 2:



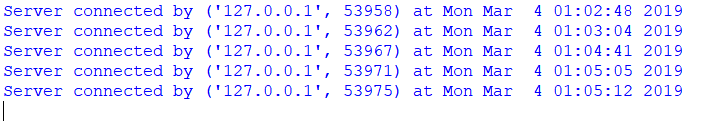
* 1. Client 3:



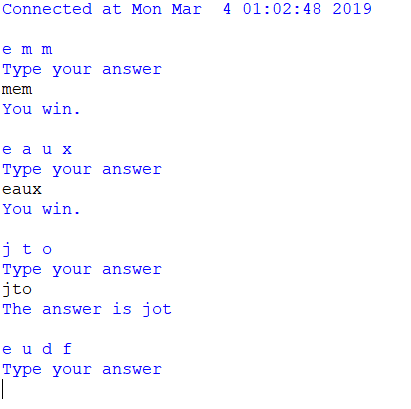
Test case #3: 5-client concurrent connection

This test case checks to see whether the server can handle concurrent connections.

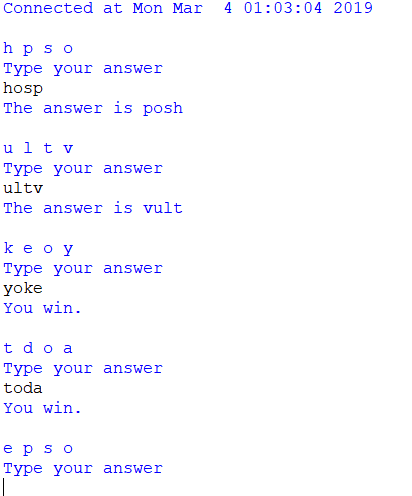
1. First, I will connect a client and solve a few jumbles. Then, I will connect 4 more clients. I will solve a few jumbles in the second client. None of the other clients should be affected.
2. I chose this test case to make sure that the server can handle multiple concurrent clients without having them interfere with each other. The words and responses should be different for all clients.
3. The outputs are as follows:
   1. Server:



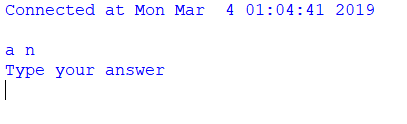
* 1. Client 1:



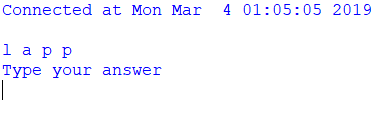
* 1. Client 2:



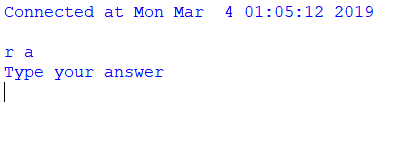
* 1. Client 3:



* 1. Client 4:



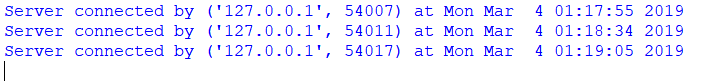
* 1. Client 5:



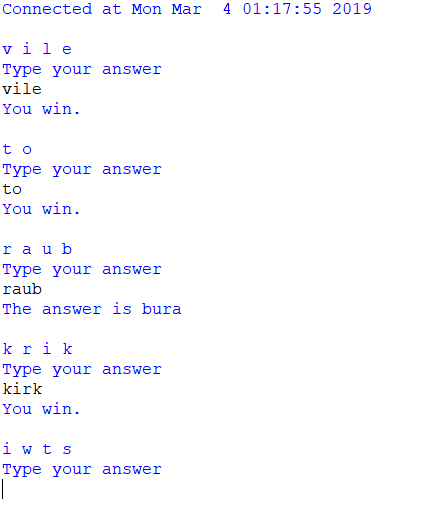
Test case #4: 3-client consecutive and concurrent connection

This test case checks to see whether the server can handle both consecutive and concurrent connections.

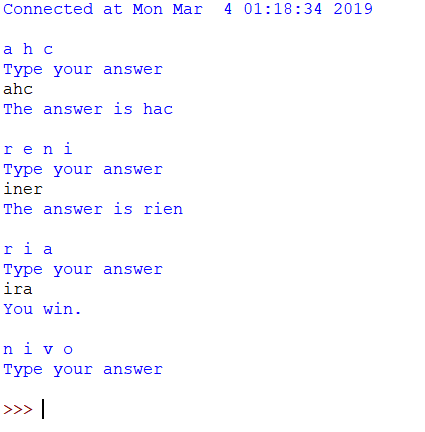
1. First, I will connect a client and solve a few jumbles. Then, I will connect a second client, solve a few jumbles, then end the connection. Finally, I will connect a third client and solve a few jumbles. Then I will go back to the first client and solve a few more.
2. This test is to make sure that the server supports both types of connections in one execution.
3. The outputs are as follows:
   1. Server:



* 1. Client 1:



* 1. Client 2:



* 1. Client 3:

