**Overall Design:**

The overall design of this project is very simple. Most of the design of the project would have went into the creation of the hangman game itself, however we were given the code for hangman to start with. The main design went into how we were going to send data over a TCP connection. My design for sending the data was pretty simple. I decided that every print that was called would be packet sent over the TCP connection from server to client, and that every input function use would be a packet sent from the client to the server. By doing this, I would be able to send information from the server to the client and back in a structured way, which will help considering a running loop will need to occur on both ends of the program and sent and received packets must be in exact amount or the program will crash. The other piece that needed to be designed was the way to not add words with non-letter characters to it. I did so by writing a simple C++ program that went through every line of the given wordlist.txt file and only print the words that didn’t have ‘ – ( and ) in them to newWordList.txt. I chose to make a new list and use that one rather than have the hangman game itself detect if the word had non-letter characters because it would reduce server processing load. Although there aren’t really many issues of server load since it can only take 5 connections, if the program were to be expanded to allow thousands to run at the same time, server load would be the primary worry since most of the processing is server based.

**How It Works:**

To start off with, the TCP connection is created between the client and server and put into a thread, which was all done by the code given to us initially. Next, the server send a greeting message to the client. Now, a loop will form in both the client and server application. The server will take the current hangman image, the letters guessed, and the word discovered so far and send it to the client. The client will print this and send an input back to the server. The server then determines if the received input is a character guessed already, if so send a message back to the server that they already guessed it. If not guessed already, then add the character to the guessed list and see if it is in the word or not, and send back a message accordingly. If it is in the word, add the letters to the so far discovered word, and if not increment the wrong count. This loop will continue until the game is over. When the game ends, the loop will break on the server side and send a message to the client according to whether they won or lost and the server will break the socket connection. Then the client program will crash after trying send anything back to the server then realizing the socket is closed. Yes, this is really bad design, but it works the same way the original hangman game and I couldn’t find a clean solution to determine if the socket has been closed before trying to receive information from it.

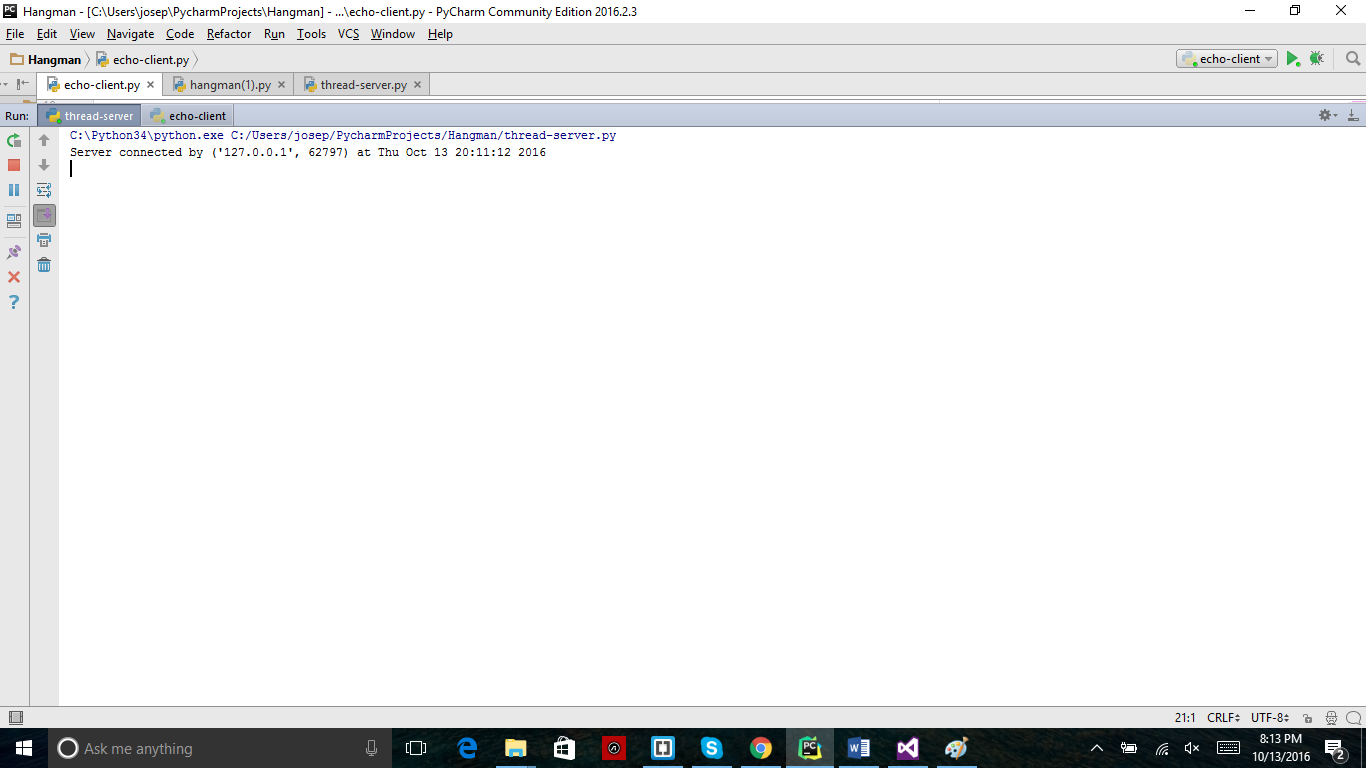
**Extensions:**

A few things can be added to this game to make it more convenient of a program. The primary thing to improve is the socket closing process. In terms of server connection, not making the client end crash on close would allow for future features to be added to the game like a retry option. Other than improving the connection closing, there isn’t too much to change about this program. It is a very straight forward game with little room for improvement in a text based format. Turning it into a GUI would makes it potentially more enjoyable.

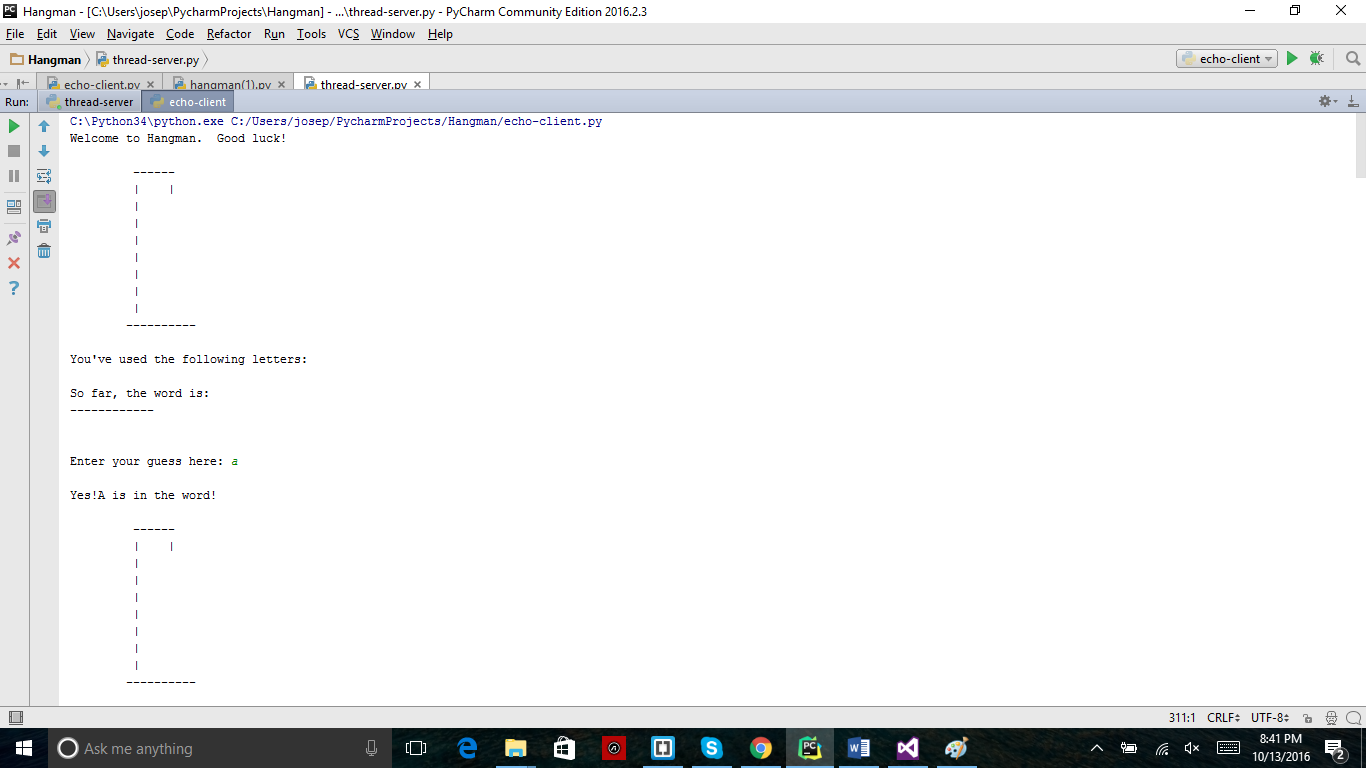
**Test Cases:**

The test I ran consisted of the following action. First, I attempted to connect the client to the server and see if the initial text was printed. Second, I tested each possible input outcome in the loop to see if they would send properly, which where character in used, character in word, and character not in word. After this, I completed the game to make sure the connection ended. Lastly, I opened multiple instances of the game to see if the server multithreading was working.

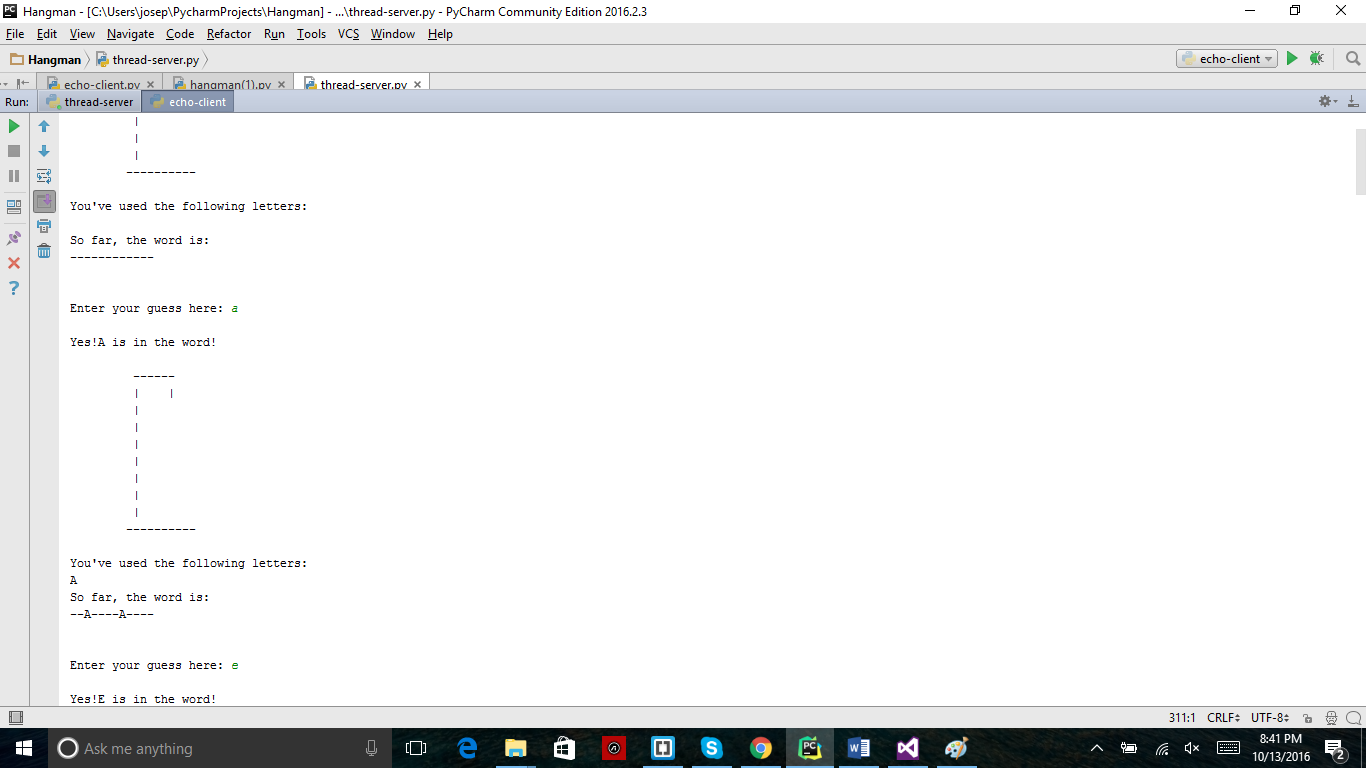
Server connected



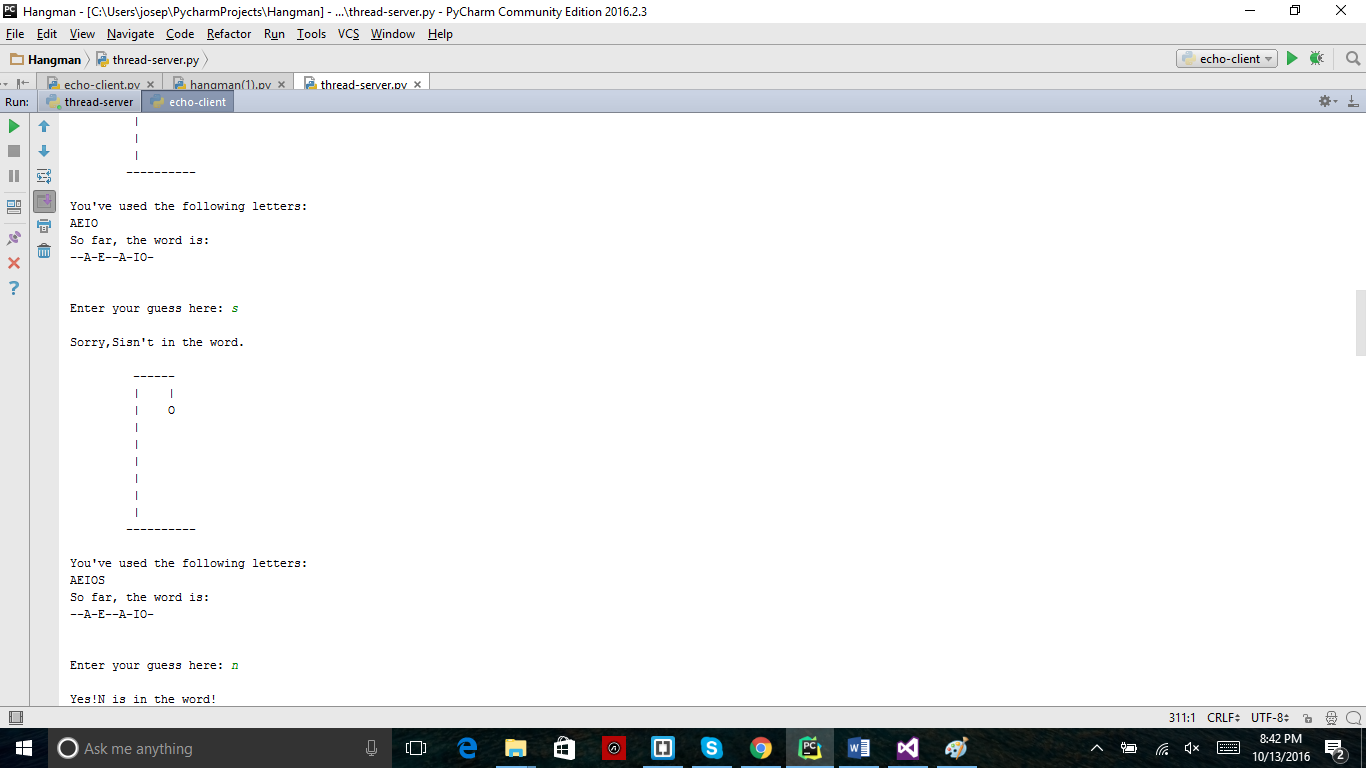
Client connected



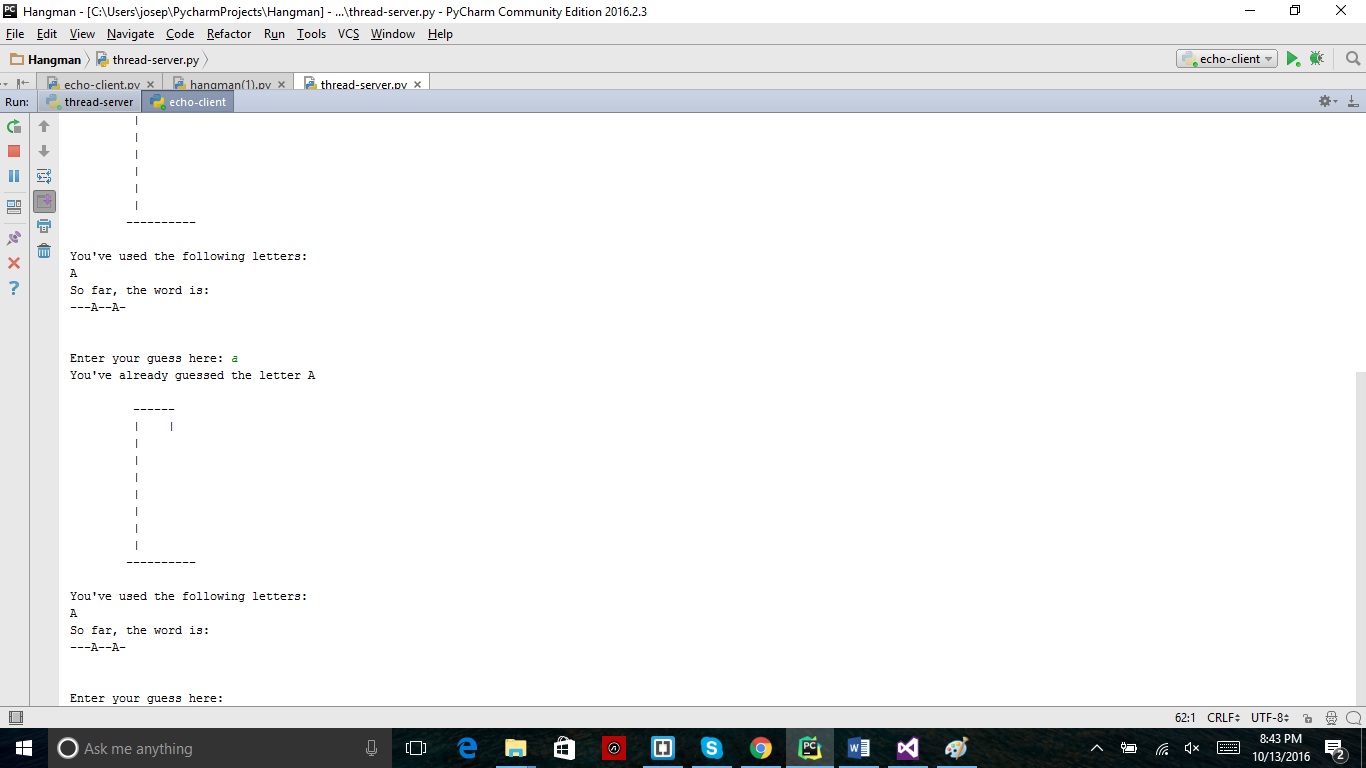
Correct guess



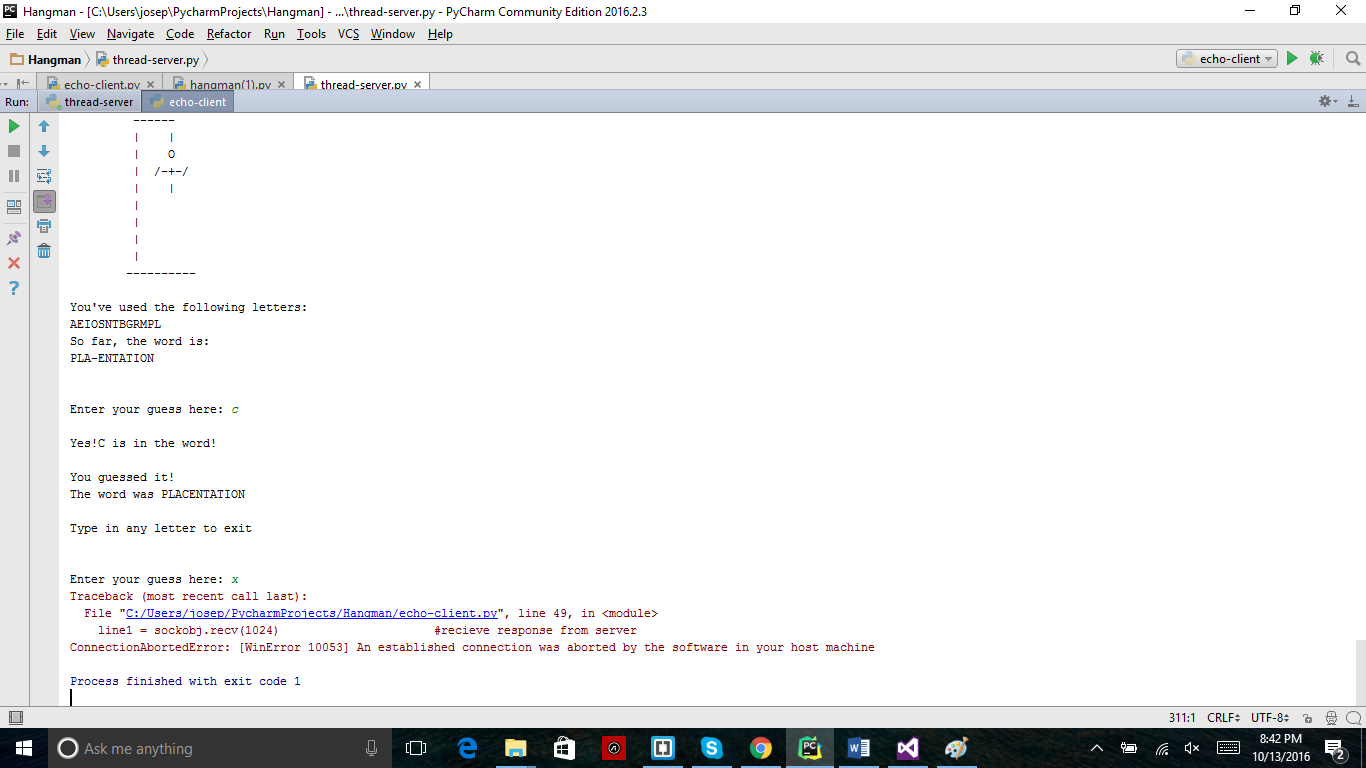
Wrong guess



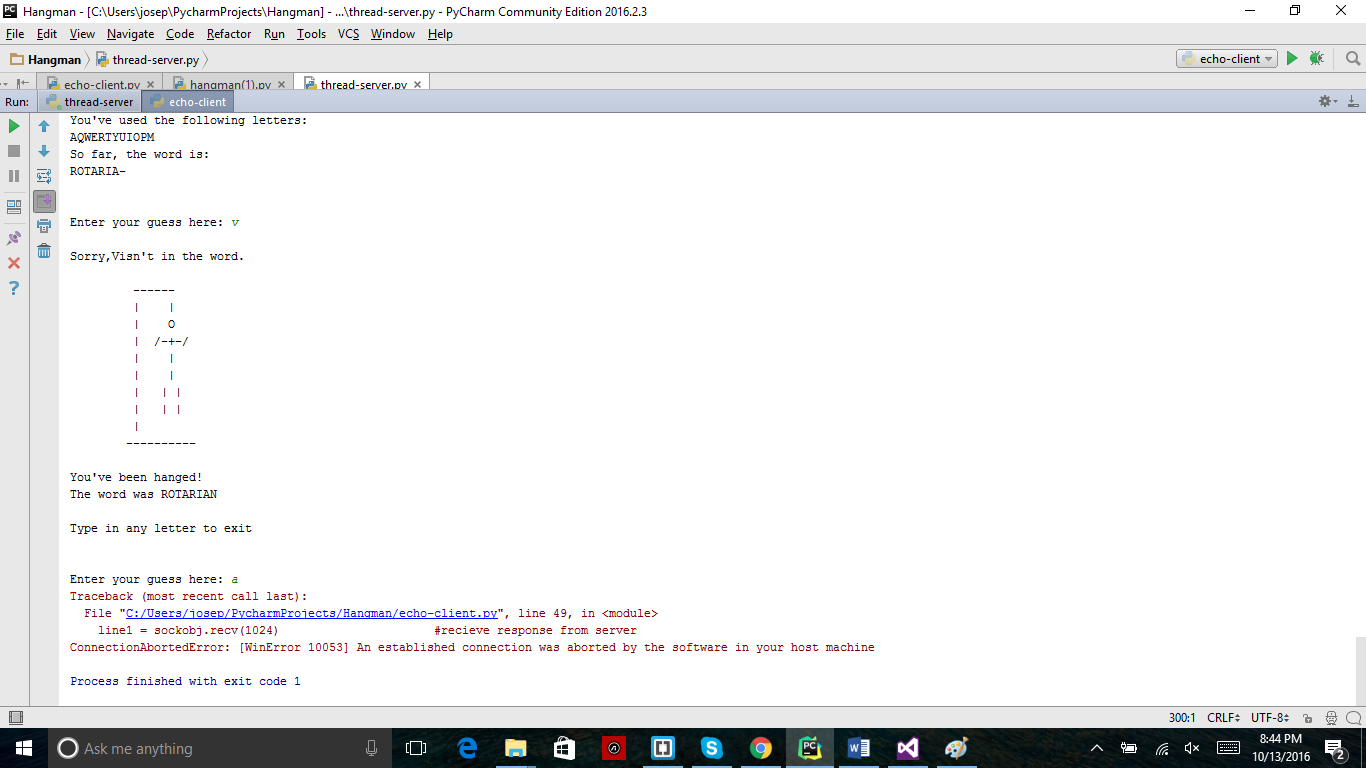
Already guessed



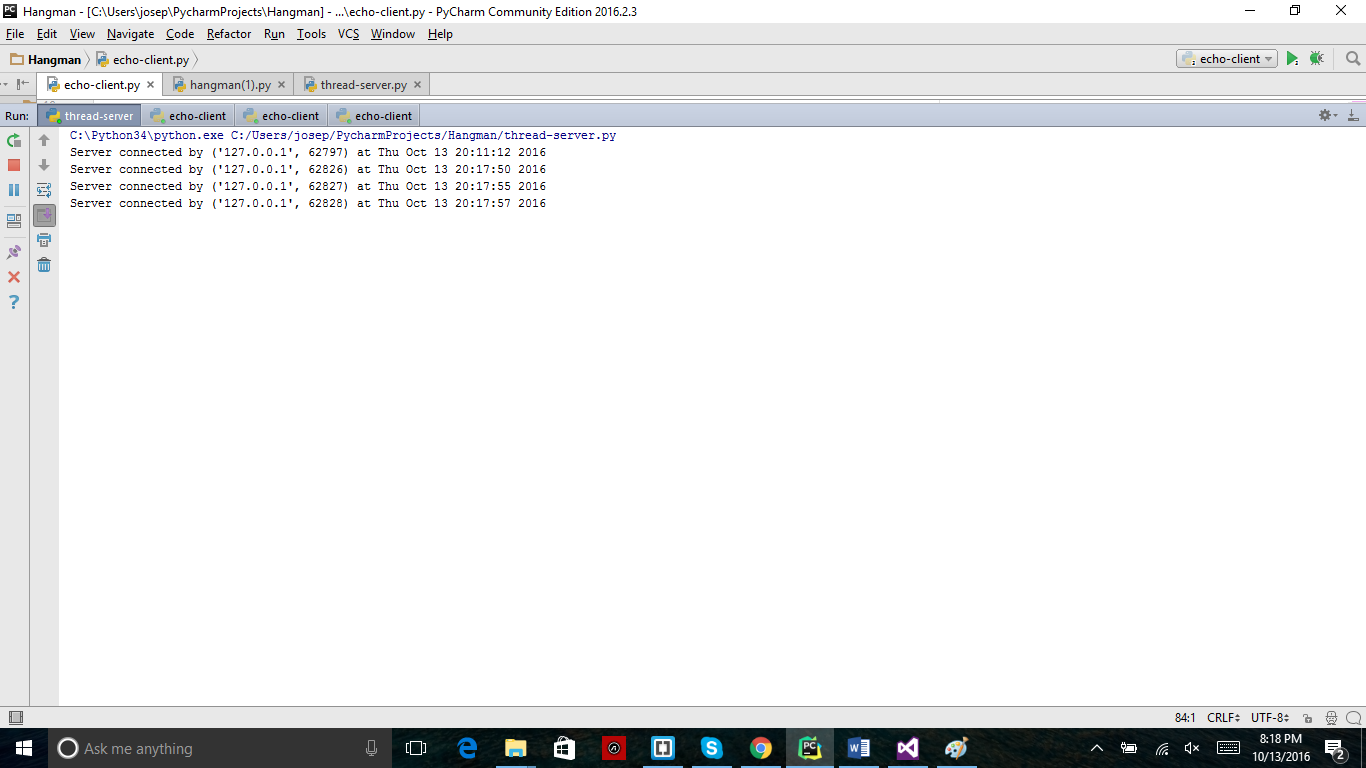
Winning



Losing



Multiple threads



All with different words

