1. Project 2: Socket Programming

This project is worth 150 points (15% of the course grade) and must be completed and turned in to the handin system before **11:59 PM on Thursday, April 26, 2018**.

2. Assignment Overview

The goal of this assignment is to gain hands-on experience programming with sockets and transport protocols (TCP and UDP). You will implement a simple game of hangman (https://en.wikipedia.org/wiki/Hangman_(game)) using socket programming in Python or C/C++ to send and receive messages between a client program and a server program. Hangman involves guessing letters of a hidden word without incorrectly guessing a specified number of times (5 in this case).

3. Project Description

The project2_skeleton.zip file is provided to you. It contains:

- Project2.pdf (This document)
- server.py
- client.py
- README.txt
- game.py (Don't modify this)
- words.txt (Don't modify this)

Your job is to complete the skeleton code in server.py and client.py to successfully play a simple game of hangman. (The hangman implementation is mostly done for you, so you need to mainly focus on sending messages back and forth using TCP and UDP sockets).

The idea is the client will contact the server by making a connection using TCP. The client will then announce itself to the server. The server responds with the UDP port the client should use for further communication. Finally, send game messages back and forth using UDP. The client repeatedly guesses letters by sending them to the server. The backend (server) handles the game logic by checking the guess and replacing the blanks with correctly guessed letters. It also decrements the number of attempts left in the case of an incorrect or already guessed letter.

Each message has 2 parts: the message type and the message text. Each message will be of the form "<message type> <message text>" (notice the space character in between). The message type will dictate to each program what to do next (this is outlined in Section 6 below). An example exchange of game messages is shown in Figure 1 for your reference.

You should modify the README.txt with your information, the machines you tested your programs on, a list of resources you used, additional comments, and some sample test output from your programs (both client and server).

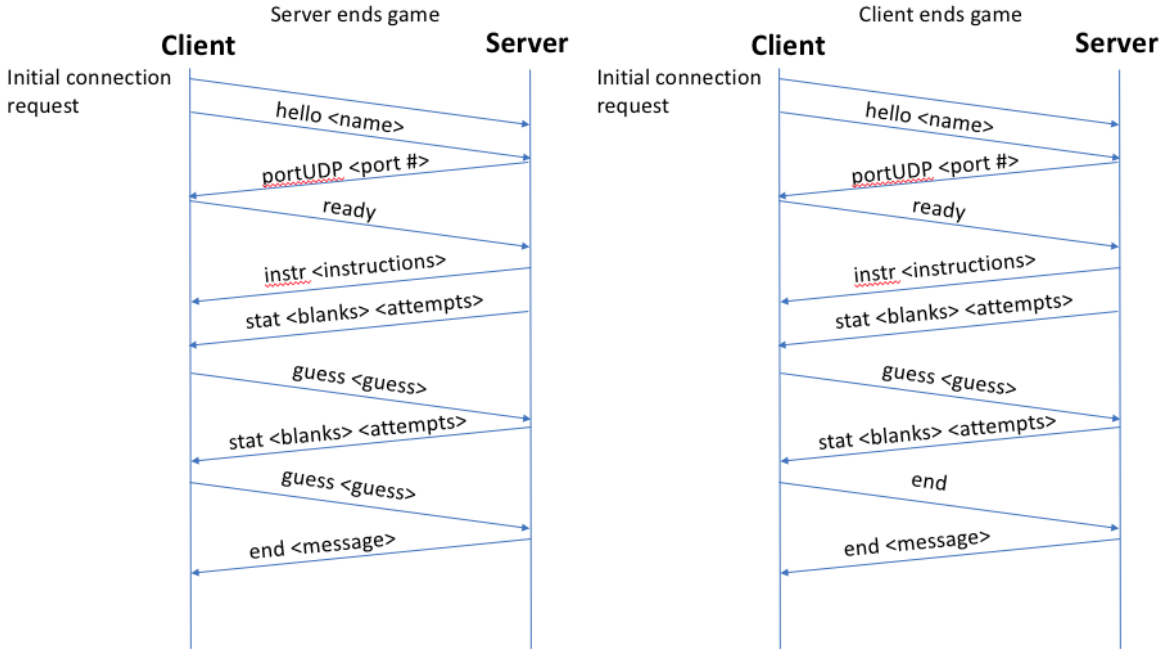

Figure 1: Game Messages Visualization

4. Client Specifications

The client should accept two command line arguments:

1. Name of the server host
2. TCP Port to be used to contact the server process

Example: python client.py arctic.cse.msu.edu 15678

When the client is first started it should prompt the user for their name. After the user has entered their name, the client should create the TCP connection to the server and send the hello message with the user's name. Once the client receives the portUDP message from the server, the client should open a UDP socket and then prompt the user and allow them to enter various game playing commands (these are different than the network messages). The client should support the game commands being entered in any case (upper, lower, or mixed case). All additional communication with the server should be performed using UDP messages to the port provided by the portUDP message.

The player (user) interacts with the client program by entering game playing commands through their keyboard. The client should support the following game playing commands:

- start – Starts a new game.
- end – Ends the current game. If a game is not already in progress this command has no effect. Sends an end message to the server indicating that the player does not want to make any more guesses. The client can start a new game or exit.
- guess <letter or word> – Used by the user to enter a guess. The user can guess either a single letter that might be part of the game word or if more than one letter is entered then the user is trying to guess the entire game word
- exit – Sends a bye message to the server, ends the client process.

Based on the game playing commands the client program sends the appropriate network message to the server to perform the actual game action. The description of these messages is shown in Section 6. That section also describes the action which should be taken when that message type is received.

5. Server Specifications

The server should accept a single command line argument. This argument should either be the option -r or a word. If the argument is -r then a client starts a new game and the word to be used for that game is randomly chosen from those contained in the supplied file words.txt. If the argument contains a word then that word will be used for all games started by a client. The server will only support one game at a time, but a client can play multiple games one right after another. The client can end a game at any time by sending an end message. The game also ends when the player either accurately guesses the word or uses all their guesses.

When started the server should create a TCP socket to be used for connections from clients. When creating this socket, it should have the system dynamically allocate an available port. The server should then display this port number. This is how clients know which port to use when contacting the server.

Examples:

python server.py -r

python server.py applejacks

After the client connects to the server, a UDP socket should be created and its port number should be sent to the client using the TCP connection. All further communication should be done using UDP messages. Section 6 describes the action which should be taken when message types are received. The server should have a timeout action on the UDP socket so that if a client disconnects or stops playing, i.e. the server does not receive a message from the client for 2 minutes (120 seconds)., the server can service other clients.

6. Messages and Message Actions

The following table lists the client and server messages that need to be used for this project. All messages consist of a "message type", a space separator, followed by message arguments. The client and server can assume that the message type is always in the specified case.

Table 1 Message Types and Descriptions

Message Type	Who sends / Receives	Description
hello	Client to Server	Used by the client to initiate a session and tell the server the player's name. Example: hello Joe
bye	Client to Server or Server to Client	Used by the client to tell the server its exiting. The server should respond by sending a corresponding bye message.
portUDP	Server to client	Used by the server to tell the client what UDP port should be used for all subsequent communications. The server when creating this port should use a dynamically allocated port. Example: portUDP 59875
ready	Client to server	Sent by the client when the user starts a new game. This message only consists of the command word "ready". It does not have any additional command arguments

Message Type	Who sends / Receives	Description
instr	Server to client	Sent by the server to provide game play instructions. Client should display the text following the message type word. Example: instr This is how you play the game.
stat	Server to client	This message has two command arguments following the command word: <blanks> and <attempts>. <blanks> is the combination of dashes and letters that represent the hidden word and <attempts> is the number of attempts the user has left to guess letters before losing.
guess	Client to server	Sent by the client when the player makes a guess. There is one message argument following the message type. This is either a single character or a full word which is the guess entered by the player. Examples: guess e guess apples
end	Server to client or client to server	When sent by the server to the client, this indicates that the server has ended the game. The client should display the message following the message type to the player, then prompt and wait for the next game command. When sent by the client to the server, this message only contains the message type. There are no message arguments. The server should end the game and respond with its own end message.
na	Server to client	Sent back to the client if the server receives an unknown message type (or possibly known message type like guess, but no game is currently started).

The server should do the following when receiving various message types:

- hello – The server now knows the name of the user. The server should send its UDP port number to the client.
- ready – Indicates to the server program that the client has a UDP port set up and is ready to start playing the game. Send the instr message followed by a stat message with the initial word blanks and attempts.
- guess – Includes one or more characters in its message text. Upon receiving a guess message, the server will use the provided checkGuess() function to update the word_blanks, attempts, and win variables. If ending conditions are met (a. guess was more than one character and it wasn't the correct word, b. there are no more attempts, or c. the player won the game), an end message will be sent with either a win or loss message text. Otherwise, if ending conditions are not met, a stat message will be sent with the updated word_blanks and attempts variables.
- end – Ends the current game. The server should respond with a corresponding end message containing a message to the player.
- bye – Ends the current game, sends a bye message back to the client, closes the udp port and TCP connection, waits for a new client TCP connection.

The client should do the following when receiving various message types:

- portUDP – Extract and store the server program's provided UDP port number. Then prompt the player for a game command.
- instr – Simply display the message text and wait to receive the initial stat message.
- stat – Display the message text then prompt for a game command from the player.
- end – Display the message text, then prompt for a game command from the player.
- na – Display the message text and then prompt for a new game command.
- bye – Display the message text and end the program.

7. Assignment Deliverables

You should turn in a .zip file to handin containing:

- server.py
- client.py
- README.txt
- game.py (non-modified)
- words.txt (non-modified)

This file should be named prj2_netid.zip where netid is your MSU network login user name. ***This is not your last name or your first name or your PID!***

8. Assignment Notes and Other Requirements

- Additional Resources:
  - Section 2-1 slides from class (from Jan 23rd)
  - Additional Socket Programming slides (from March 22nd)
  - Python Socket documentation (https://docs.python.org/3.6/library/socket.html)

- The use of encode() and decode() methods should be used for converting messages between Unicode and byte representation
- The Python split() method "splits" a string into a list with the space character as the default delimiter
    - Use try/except for cases where list indices may cause an IndexError
- Remember to close sockets at the end of each program
- To test your program, use any of the CSE servers that are accessible via SSH (arctic, black, or any of the Raspberry Pis)
    - http://www.cse.msu.edu/Facility/Services/SSH.php
    - http://www.cse.msu.edu/pie/index.html
    - Open two separate connections (one to run your client program and one to run your server program).
- Use try/except to catch errors whenever possible. (i.e. timeouts, ctrl-c keyboard interrupt, unexpected IndexError)

9. Grading Rubric

General

__ / 10 Proper completion of README.txt

Client Program (70 Points)

__ / 15 TCP Port setup successfully

__ / 15 UDP Port setup successfully

__ / 10 Properly handled portUDP message

__ / 10 Properly handled the instr message

__ / 10 Properly handled stat message

__ / 10 Properly handled the end message

Server Program (70 Points)

__ / 15 TCP Port setup successfully

__ / 15 UDP Port setup successfully

__ / 10 Properly handled ready message

__ / 30 Properly handled the guess message

10. Sample Client and server game output.

Exchange 1

Server Output

```
>python3 jf_server.py -r
Server is running...
Creating TCP socket...
TCP socket has port number: 46491
Waiting for a client...
A new client is connected to the server!
User's name: Dennis
Creating UDP socket...
UDP socket has port number: 39256
Sending UDP port number to client using TCP connection...
Hidden Word: also
Starting game...
Sending message: stat Word: ---- Attempts left: 5
Guess is: e
Incorrectly or already guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: ---- Attempts left: 4
Guess is: a
Correctly guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: a--- Attempts left: 4
Guess is: o
Correctly guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: a--o Attempts left: 4
Guess is: l
Correctly guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: al-o Attempts left: 4
Guess is: s
Correctly guessed char
Attempts left: 4
Correctly guessed word
Win status: True
Sending message: end You win! Word was also
Hidden Word: wife
Starting game...
Sending message: stat Word: ---- Attempts left: 5
Guess is: e
Correctly guessed char
Attempts left: 5
Win status: False
Sending message: stat Word: ---e Attempts left: 5
Guess is: o
Incorrectly or already guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: ---e Attempts left: 4
Guess is: o
Incorrectly or already guessed char
Attempts left: 3
Win status: False
Sending message: stat Word: ---e Attempts left: 3
Guess is: i
```

Server Output

```
Correctly guessed char
Attempts left: 3
Win status: False
Sending message: stat Word: -i-e Attempts left: 3
Guess is: f
Correctly guessed char
Attempts left: 3
Win status: False
Sending message: stat Word: -ife Attempts left: 3
Guess is: w
Correctly guessed char
Attempts left: 3
Correctly guessed word
Win status: True
Sending message: end You win! Word was wife
Client exiting
Waiting for a client...
^CClosing TCP and UDP sockets...
```

Client Output

```
>python3 jf_client.py localhost 46491
Client is running...
Remote host: localhost, remote TCP port: 46491
Please enter your name: Dennis
Server IP: 127.0.0.1
Connected to server!
Reading TCP socket for UDP Port info...
UDP Port on Server: 39256
> start
This is hangman. You will guess one letter at a time. If the letter is in the hidden word, the "-
" will be replaced by the correct letter. Guessing multiple letters at a time will be considered
as guessing the entire word (which will result in either a win or loss automatically - win if
correct, loss if incorrect). You win if you either guess all of the correct letters or guess the
word correctly. You lose if you run out of attempts. Attempts will be decremented in the case of
an incorrect or repeated letter guess.
Word: ---- Attempts left: 5
> guess e
Word: ---- Attempts left: 4
> guess a
Word: a--- Attempts left: 4
> guess o
Word: a--o Attempts left: 4
> guess l
Word: al-o Attempts left: 4
> guess s
You win! Word was also
> start
This is hangman. You will guess one letter at a time. If the letter is in the hidden word, the "-
" will be replaced by the correct letter. Guessing multiple letters at a time will be considered
as guessing the entire word (which will result in either a win or loss automatically - win if
correct, loss if incorrect). You win if you either guess all of the correct letters or guess the
word correctly. You lose if you run out of attempts. Attempts will be decremented in the case of
an incorrect or repeated letter guess.
Word: ---- Attempts left: 5
> guess e
Word: ---e Attempts left: 5
> guess o
Word: ---e Attempts left: 4
```

Client Output

```
> guess o
Word: ---e Attempts left: 3
> guess i
Word: -i-e Attempts left: 3
> guess f
Word: -ife Attempts left: 3
> guess w
You win! Word was wife
> exit

Closing TCP and UDP sockets...
```

Exchange 2

Server Output

```
>python3 jf_server.py -r
Server is running...
Creating TCP socket...
TCP socket has port number: 54281
Waiting for a client...
A new client is connected to the server!
User's name: Dennis
Creating UDP socket...
UDP socket has port number: 56163
Sending UDP port number to client using TCP connection...
Hidden Word: spoke
Starting game...
Sending message: stat Word: ----- Attempts left: 5
Guess is: e
Correctly guessed char
Attempts left: 5
Win status: False
Sending message: stat Word: ----e Attempts left: 5
Guess is: i
Incorrectly or already guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: ----e Attempts left: 4
Guess is: o
Correctly guessed char
Attempts left: 4
Win status: False
Sending message: stat Word: --o-e Attempts left: 4
Guess is: t
Incorrectly or already guessed char
Attempts left: 3
Win status: False
Sending message: stat Word: --o-e Attempts left: 3
Guess is: s
Correctly guessed char
Attempts left: 3
Win status: False
Sending message: stat Word: s-o-e Attempts left: 3
Guess is: l
Incorrectly or already guessed char
Attempts left: 2
Win status: False
Sending message: stat Word: s-o-e Attempts left: 2
Guess is: p
Correctly guessed char
Attempts left: 2
Win status: False
```

Server Output

```
Sending message: stat Word: spo-e Attempts left: 2
Guess is: f
Incorrectly or already guessed char
Attempts left: 1
Win status: False
Sending message: stat Word: spo-e Attempts left: 1
Guess is: w
Incorrectly or already guessed char
Attempts left: 0
Win status: False
Sending message: end You lose! Word was spoke
Client exiting
Waiting for a client...
^CClosing TCP and UDP sockets...
```

Client Output

```
>python3 jf_client.py localhost 54281
Client is running...
Remote host: localhost, remote TCP port: 54281
Please enter your name: Dennis
Server IP: 127.0.0.1
Connected to server!
Reading TCP socket for UDP Port info...
UDP Port on Server: 56163
> start
This is hangman. You will guess one letter at a time. If the letter is in the hidden word, the "-
" will be replaced by the correct letter. Guessing multiple letters at a time will be considered
as guessing the entire word (which will result in either a win or loss automatically - win if
correct, loss if incorrect). You win if you either guess all of the correct letters or guess the
word correctly. You lose if you run out of attempts. Attempts will be decremented in the case of
an incorrect or repeated letter guess.
Word: ----- Attempts left: 5
> guess e
Word: ----e Attempts left: 5
> guess i
Word: ----e Attempts left: 4
> guess o
Word: --o-e Attempts left: 4
> guess t
Word: --o-e Attempts left: 3
> guess s
Word: s-o-e Attempts left: 3
> guess l
Word: s-o-e Attempts left: 2
> guess p
Word: spo-e Attempts left: 2
> guess f
Word: spo-e Attempts left: 1
> guess w
You lose! Word was spoke
> bye
Invalid command. Try again.
> exit

Closing TCP and UDP sockets...
```