INTRODUCTION

The second project of the networking lab will be to create a server application that ECHOs a command that is sent from a custom client application. These ECHO client and server applications will interact across a TCP/IP-based network using a simple request/response Application-layer protocol.

BACKGROUND

With Project 1, we built an application that prompts for, and processes input strings, reads a configuration file, and writes entries to a log file to show processing status as well as warnings and errors. These functions will be important in Project 2.

Project 2 will extend the functionality of Project 1 to serve as the custom client to a new server application which we will write. The new server application that will take requests from the client application and return appropriate responses.

The concepts that will be introduced in this project will be:

- Using the socket Interface for both the client and server applications
- Constructing and parsing application layer requests and responses using JavaScript Object Notation (JSON)
- Passing messages between client and server applications using a TCP/IP-based network

THE LOCAL LOOP

In this project, we will make use of the local loop network to make connections between the client and server applications. This is due to security limitations in the OSU Student Linux environment. In “real world” situations, using the local loop is an appropriate way to test networked applications before deployment onto the corporate network or the Internet.

BEFORE STARTING

Before you jump into this project, please consider making a backup of your Project 1 source code just in case something would go wrong. Also consider using a software repository, like GitHub, to store your source code and organize your projects.

REQUIREMENTS

CLIENT APPLICATION

Create a standalone, interactive, and fully commented client application that interacts with a user and serves as an interface to a server application and performs these tasks:

1. Open and process a configuration file (see below for details)
2. Open a log file using parameters (filename, logging level, and file mode) obtained from the configuration file and write all major activity to the log file
3. Connect to the server application over the network using the IP address and port number from the configuration file

   a. Upon your connection being successful, write an informational "Connected to …" message to the screen

4. Loop while prompting the user of the application for input strings.
   a. If an empty string was entered, print an appropriate error reporting that no command was entered
5. Process each input string as follows:
   a. Write the original input string to the log file
   b. Parse the input string by tokenizing the string, delimiting on whitespace (blanks)
   c. Convert (translate) only the first token to all uppercase characters. The first token will be the ***command*** in the ***server request***
   d. Concatenate the remainder of the tokens into a string, each separated by whitespace (blanks). This string will be the ***parameter*** in the ***server request*** (Note: the parameter string may be empty)
6. Create a ***server request*** from the ***command*** and the ***parameter*** as a JSON-formatted string (see details below)
7. Send the ***server request*** to the server application and wait for a ***server response***
   a. The ***server response*** will consist of a ***response*** element and a ***parameter*** element (see Server Application requirements below)
8. Read the ***server response*** and:
   a. Write the entire received ***server response*** to the log file
   b. Print the ***parameter*** element of the ***server response*** received to the screen
   c. If the ***response*** element in the ***server response*** is "QUITTING":
      i. Print an appropriate "Shutting down" message to the screen and log file
      ii. Exit the application

Client Application Details

1. Input strings should be made up of one or more "words" (groups of alphanumeric characters)
2. Copy the Project1 configuration file to create a client configuration file
3. In the client configuration file, If the sections below do not exist, add them; if they do exist, alter them as follows:

```
[project2]
# server IP address
serverHost = 127.0.0.1

# server listening port (> 1024)
serverPort = 7721

[logging]
# log file filename
```

```
logFile = <username.#>-project2-client.log

# logging level
logLevel = INFO

# log file mode (w = write, a = append)
logFileMode = a
```

4. From the configuration file, **`project2`** section, read the ***`serverHost`*** and ***`serverPort`*** configuration fields and store their parameter values.
5. From the configuration file, **`logging`** section, read the ***`logFile`***, ***`logLevel`***, and ***`logFileMode`*** configuration fields and store their parameter values
6. When creating a socket object, use the **`AF_INET`** address family (so host can be expressed as a hostname) and the **`SOCK_STREAM`** socket type (TCP socket).
7. When connecting to the server, the server address is passed as a tuple containing the hostname and port number. Note: This is the "3-way handshake" since we're using TCP.
8. The ***server request*** format should be a JSON-formatted string in this form:

```
{
    "command" : "<command>",
    "parameter" : "<string>"
}
```

SERVER APPLICATION

Create a standalone and fully commented server application that performs these tasks:

1. Open and process a configuration file
2. Open a log file using parameters (filename, logging level, and file mode) obtained from the configuration file and write all major activity to the log file
3. Open a socket and bind to the IP address and port specified by the ***serverHost*** ***`serverPort`*** configuration options
4. Listen for an incoming client connection. When one arrives, accept it
    a. Upon accepting the connection, write an appropriate "Connection accepted from …" to the log file
5. Loop while processing ***server request***s
6. Read the ***server request*** from the network
7. Parse the ***server request***, saving the ***command*** and ***parameter*** in separate variables
    a. If the client request is "QUIT"
        i. Format a ***server response*** with "QUITTING" in the ***response*** element and an empty (null) ***parameter*** element
        ii. Send the ***server response*** to the client
        iii. Close the client connection

iv. Exit the server application gracefully

b. If the client request is not "QUIT"

i. Format a ***server response*** with the ***response*** element set the same as the ***command*** element from the ***server request***

ii. Set the ***parameter*** element of the ***server response*** to "Server received: " followed by contents of the ***parameter*** element from the ***server request***

iii. Send ***server response*** to the client

Server Application Details

1. Copy the client configuration file to create a server configuration file.
2. Change the logFile option to indicate it will be the server log:

```
logFile = <username.#>-project2-server.log
```

3. From the configuration file, **`project2`** section, read the ***`serverHost`*** and ***`serverPort`*** configuration fields and store their parameter values.
4. From the configuration file, **`logging`** section, read the ***`logFile`***, ***`logLevel`***, and ***`logFileMode`*** configuration fields and store their parameter values
5. When creating a socket object, use the **`AF_INET`** address family (so host can be expressed as a hostname) and the **`SOCK_STREAM`** socket type (TCP socket).
6. The server response will be returned in a JSON-formatted string in this form:

```
{
    "response" : "<response>",
    "parameter" : "<string>"
}
```

GRADING

To earn credit for this project, you will need to;

1. Record a video of you running your completed project in the OSU Student Linux environment
2. Submitting the video, along with your source code and the log files produced after running your server and client applications, to Carmen

STEPS

1. Login to the OSU Student Linux environment twice, with two terminal sessions
2. Upload your application (Python script) and the provided configuration file
3. Begin recording
4. Run your server application (using **`python3`**) in one session

5. Run your client application (using **`python3`**) in the other session and type in each of these commands:
   a. jumped over the lazy dog
   b. call Call me Ishmael
   c. In a hole in the ground there lived a hobbit
   d. houston, we have a problem
   e. it was a dark and stormy night
   f. i want to quit
   g. quitting
   h. quit
6. After both of your applications have stopped, stop recording
7. Download the log files and configuration files from both your client and server applications
8. Create a ZIP file with your source code, configuration files, log files, and the recording of your session
9. Submit your ZIP file to Carmen

CREDIT

Credit will be given as follows:

Client Application

• Client application runs without failures/errors	5 Points
• Client application fulfills each of the stated requirements	10 Points

Server Application

• Server application runs without failures/errors	5 points
• Server application fulfills each of the stated requirements	10 Points

Partial credit will be given wherever possible.

NOTES

- Python resources:
  - https://realpython.com/python-sockets
  - https://pythontic.com/modules/socket/client-server-example
  - https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client
- Commenting code is an industry best practice and will assist in assigning partial credit if necessary
- Please review the Grading section of the syllabus