EE 3233 Systems Programming for Engineers (Fall 2023)

Final Project

Name: Elijah Guzman Score:_____/120

For the final project you will build a remote access tool (RAT). A remote Access Tool is used to remotely access or control a computer. There are two main components to a RAT, the server and the client(s). The final can be done in Python or C. (much easier in python).
It seems like a lot of work but there is some code reuse and it's not that bad once you start.

- **Server (60 pts)**– The server takes input from a user and sends instructions to a client and requests specific actions such as:
    - Run a command
    - Receive a file (download file)
    - Send a file (upload file)

- **Client (60pts)** – The client receives instructions from the server and performs instructions such as:
    - Command execution
    - Send a file (upload file)
    - Receive a file (download file)

Due date: Friday, 08th of the Dec, 2023 by 11:59 PM

Implement the following core modules/functions (I'll provide some starter code): This is majority of points and it can be used in both the server and client.

1. **Command**
    a. Python instructions
        i. Write a module **command.py.** This will have a class called **Command** and function called **run_command** that uses Popen to run a user provided command (similar to recent lab).
    b. C instructions
        i. Write a new file called **command.c.** This will have a function that calls popen.
2. **FileIO**
    a. Python instructions
        i. Implement a module called **fileio.py.** Create a class called **FileIO** that will have two functions **write_file** that writes given input to a file and **read_file** that reads a file and returns the data.
    b. C instructions
        i. Write a new file called **fileio.c**. You'll have two functions **read_file** for reading from a specific file and returning the data, and **write_file** to write the given data to a specific file

Now that the core is written, you'll write the **server.py** or **server.c**.

1. The server side is the standard setup.
    a. Socket
    b. Bind
    c. Listen
    d. Accept
2. After a successful connection will be a series of if/else commands to match the functionality of the RAT (run_command, upload_file, download_file)
    a. **run_command:** you have the code already. Call your command module/functions and send the results.
    b. **upload_file**: This is already done as well! If the server is uploading a file to the client. It has to read the file first (You have your **FileIO** module). Read the file and send the data.
    c. **download_file:** Similar to upload_file, however, you'll receive a stream of bytes from the client, now you use **FileIO** module, but you write to disk instead of reading. (just pick a default name of "received_file.txt" to make it easier)
    d. Implement **exit** if the client requests to end the connection.

3. BONUS: (Recommend backing up solutions before attempting bonus)
    a. +5 Adapt your server handle multiple clients using multi-threading
    b. +5 Adapt your server to handle dynamic filenames. This requires additional input/output from the user.

3. Write an interactive **client** (**client.py OR client.c)** that takes user input to perform an action on the server.

```
$ python3 client.py
Enter a command to perform
 run_command
 upload_file
 download_file
```

    a. **run_command**: After the user enters run_command. The server should prompt what command should be run on the client.

```
$ python3 client.py
Enter a command to perform
 run_command
 upload_file
 download_file
run_command
Server: What command would you like to run?
ls -alh
```

Send “ls -alh” or whichever command the user enters on the keyboard and send the command to the server.

b. **upload_file:** Use your **FIleIO** module to read the contents of a file. Then send that data to the server. On the server side, you would call your **download_file** function giving it the data sent from the client (because if the client is uploading, the server is downloading).
    a. If you’re doing the extra credit, you may want to prompt for the name of the file and send that before you send the file. If not doing the bonus, use the default filename

```
$ python3 client.py
Enter a command to perform
 run_command
 upload_file
 download_file
upload_file
Server: Waiting for file
Client: Sending file
Server: File Received
```

c. **download_file:** Downloading a file is receiving data from the server and using your **FileIO** module and writing it to a file.
    a. If doing the bonus, you may want to prompt the user for the name of the file to download. If not doing the bonus, just use the default filename

```
$ python3 client.py
Enter a command to perform
 run_command
 upload_file
 download_file
download_file
Client: File received
```

d. **exit**: After the user enters exits. Close the connection

*NOTE: It’s easier to send/receive in order, vs doing combinations of like send, send, receive, receive.

MORE BONUS:
There's a server running **IP: <TBD>, PORT 8080**
Make a copy of your client and see if you can modify a copy to connect to the server and find some flags for extra bonus points. (25 pts total)

The server implements **run_command, upload_file, download_file**

1. Try running some commands to discover what's present ("ls -alh" or "ps -aux"). There are 2 flags you can get (md5 hashes), and a file.
2. Send a file with your name inside the file for additional points