Computer Engineering 4DN4
Laboratory 2
Online Grade Retrieval Application

The objective of this lab is to develop a client/server network application that can be used by course instructors to distribute grades. The code will be written in Python 3 using the Berkeley/POSIX socket API discussed in class. The server software would be run by the course instructor and is given access to a database of student grade records. The client software communicates with the server in order to retrieve and verify student course grade information. The client must authenticate itself before course grade data can be retrieved. A client can also issue commands that will return the requested grade. The working system has to be demonstrated by each team either in person (in BSB 238) or online (via MS Teams). Sign-up sheets will be available later for each team to choose their demo time and format.

1 Description

The software to be developed consists of separate server and client applications written using Python 3. The required functionality is described as follows.

1.1 Server

The server code is run from the command line on a server host and awaits TCP connections from clients. If the right conditions are satisfied, the server responds with student grade information, which is then displayed on the client console. The details are as follows.

1. A mark database is provided to the server in the form of a CSV (comma-separated values) file. A CSV file can be exported from a standard Microsoft Excel spreadsheet file by using the `File` menu and selecting `Save As --> CSV`. An example of a CSV file named `course_grades.csv` is posted with this lab including the Excel spreadsheet, `course_grades.xlsx`, from which it was obtained.
2. When the server is started, it reads in the CSV file and listens for client TCP connections. An example of reading and writing a CSV file is given on the course homepage on Avenue. This is available under the "Content --> Reviews --> Python Exercises". Please check how `company.py` reads the database file. An example CSV file is shown below, where the first row defines the meaning of the columns, and the first two entries in each row are the student's ID number and password.

```
ID Number,Password,Last Name,First Name,Midterm,Lab 1,Lab 2,Lab 3,Lab 4
1788788,SiKoLkVb,Smith,Thomas,43,47,74,84,63
1769647,NyKzKzMe,Murphy,William,89,61,81,77,69
1770270,LcEsHzQh,Lam,Nathan,86,41,100,74,46
1792727,VuOiRpSk,Martin,Judy,90,40,40,97,52
1706159,YcYjNtLa,Brown,Jane,52,75,64,81,56
1742218,SwAcVmHg,Roy,Jim,81,56,67,64,98
1741498,PmZxLpEg,Temblay,Alex,98,97,51,86,75
1701464,KtPbIiMg,Lee,Noah,72,46,77,48,62
1789251,KkQyDbZv,Gagnon,Felix,92,77,92,80,43
1706475,XzYtMfZc,Wilson,Oliver,47,92,48,96,91
```

3. When a client connection occurs, the server reads what has been sent from the client. There are five commands that can be received, i.e., GMA, GL1A, GL2A, GL3A, and GL4A, where GMA is "get midterm average" and the others are "get lab average" commands, e.g., GL3A is "get lab 3 average", and so on. When one of the "get average" commands is received, the server computes the appropriate grade average, then returns the value to the client, where it is displayed on the console. The commands can be defined as string constants in your program, e.g.,

   ```
   GET_MIDTERM_AVG_CMD = "GMA"
   GET_LAB_1_AVG_CMD = "GL1A"
   GET_LAB_2_AVG_CMD = "GL2A"
   ...
   ```

   If the decoded bytes from the client is not one of the above commands, then the server interprets it as a hashed ID/password that is used to authenticate the identity of the client request. The input is compared to each of the hashed ID/passwords of the CSV file records to see if there is a match. A student grade record will only be sent to the client if a proper password authentication is made, If an improper request is made, the server returns an error message to the client. More details of client and server behaviour are as follows.

1.2 Client

The client code is run on a client host and creates a TCP connection to access the server mark database. The details are as follows.

1. The client runs as a command line application and first prompts the user for an input command. If the user enters one of the "get averages" commands, then the client creates a TCP connection to the server and sends the command over the connection. The response from the server is then printed on the command line.

2. In addition to recognizing the "get averages" commands, the client also recognizes a "get grades" command, i.e., "GG". When this command is entered, then the client assumes that the user would like to retrieve his/her grades. In order to retrieve student marks, the client

must authenticate itself to the server. This is to ensure that a student cannot easily determine the grades of another student.

The authentication works as follows. The client first prompts the user for his/her student ID number and password using the Python input function and the getpass module. See

`https://docs.python.org/3.1/library/getpass.html`

for some simple examples. The client then creates a TCP connection to the server and sends a message consisting of a secure hash of the entered ID and password. This can be easily obtained using the Python hashlib module and using the sha256 hash. The full hash can be obtained by first creating the hash object and then calling the hashlib update() function twice, first giving it the student's ID number and then the password. The ID/password hash is then obtained by calling the digest() function. Details of doing this can be found at

`https://docs.python.org/3/library/hashlib.html`,

which includes some simple examples. Note that the ID and password strings have to be encoded into bytes objects before you can call the hashlib functions. You can use utf-8 encoding to accomplish this, e.g., `"my password".encode("utf-8")`.[1]

3. Once the ID/password hash has been checked by the server, it responds with the record data associated with that student or returns an error message. You need to interpret it and print it out on the screen properly. After the printout, the client should close the connection and return to the command prompt state. At that point, the client should be able to reconnect to the server without restarting either application.

2 Requirements

Teams: You can work in teams of up to 3 students.

Demonstration: This can be done in ONE of the following ways.

(1) In-person in room BSB 238. All team members must attend the demonstration and bring own laptops. The laptops must be connected to the `COE4DN4` Wi-Fi (password `IPv4socket`). The working system must be demonstrated on two of the team's laptops, with one acting as the server and the other as the client. For teams of 3, the TA may ask the server or client to switch to the third laptop at any time during the demonstration.

[1] In a practical system one should never concoct ones own ad hoc security mechanisms. Instead, libraries should be used that have been created for this purpose. The method we are using here, for example, has some insecurities. A replay attack could be used, for example, where, if someone steals (or intercepts) a student's hashed password, could then connect to the server and use it to retrieve the student's grade record. It does, however, provide protection of the identity of the student, since the hash is difficult to reverse. That is why, in this application, it is best for the server to return only the student's grades, without divulging the name or ID number of the student. A replay attacker could still retrieve the grades, but could not associate them with a particular student. In practice, passwords would never be stored in plain text at the server. Instead, a salted hash of each password would usually be stored. In a production system it is best to use authentication mechanisms that are acknowledged to be secure.

(2) Online via MS Teams. All team members must attend the demonstration. During the demonstration, one member must share their screen and run both the server and client on the same computer. The TA may ask a different member to continue or restart the demo at any time during the demonstration.

Each team needs to reserve a 15-minute time slot in one of the designated lab sessions, either online or in-person. Time slots will be assigned on a first-come-first-served basis using the Doodle scheduler (links will be posted on Avenue later). All team members have to show up. Make sure you are certain of the team's schedule, since once a time slot is booked, it cannot be changed.

Marking Scheme: The assigned mark consists of two parts:

- 80% for demonstration, and
- 20% for implementation.

Demonstration: The demonstration consists of the following, where each step below is weighted equally in the demonstration component of the mark.

1. Start the server in a shell window. The server prints out the data read from the CSV file, e.g., "Data read from CSV file: ", followed by each row of the file.
2. The server must print output indicating that it is listening on the host laptop for incoming connections on a particular TCP port, e.g., "Listening for connections on port `<port number>`."
3. Start the client in another shell window of the SAME computer. The client will prompt the user for a command. When a command is entered, the client echos what has been entered, e.g., "Command entered: `<cmd>`".
4. If the command entered is a "get average" command, then the client will output a message such as"Fetching Lab 1 average:". This is followed by the average returned by the server.
5. If the command entered is "GG", the user is prompted for his/her student ID number. Once an ID number is input, the client prompts the user for a password. The client must then print out the values that were input, e.g., "ID number `<ID number>` and password `<password>` received."
6. The client then outputs the hash sent to the server, e.g., "ID/password hash `<hash value>` sent to server."
7. When the server receives the TCP connection, it should print output, e.g., "Connection received from `<IP address>` on port `<port>`."
8. The server should print out what it has received, e.g., "Received GL3A command from client." or "Received IP/password hash `<hash value>` from client."
9. If an ID/password was sent and matches a database entry, the server outputs a confirmation message, e.g., "Correct password, record found." Otherwise, the server prints out an error message, e.g., "Password failure." The correct database record or failure message is then returned to the client.

10. The client will then retrieve the server response and output it on the terminal window.
11. Steps 3 to 10, above, should be repeatable without restarting the client or server.

Implementation:

1. Your Python code and related description. The description should clearly explain how you constructed the code to achieve the required functionality.
2. List three mistakes (excluding syntax errors) that you made during the implementation process. Describe your observations and briefly explain how they led you to identify and fix each mistake.

Writeup: .

1. A PDF report that briefly describes how you implemented the client and server applications, including how you fixed the selected mistakes.
2. A single Python source code (py) file that includes both the client and server classes that you created.