IT assignment 1

Soumalya Kundu
Roll: 001810501033

Goals:

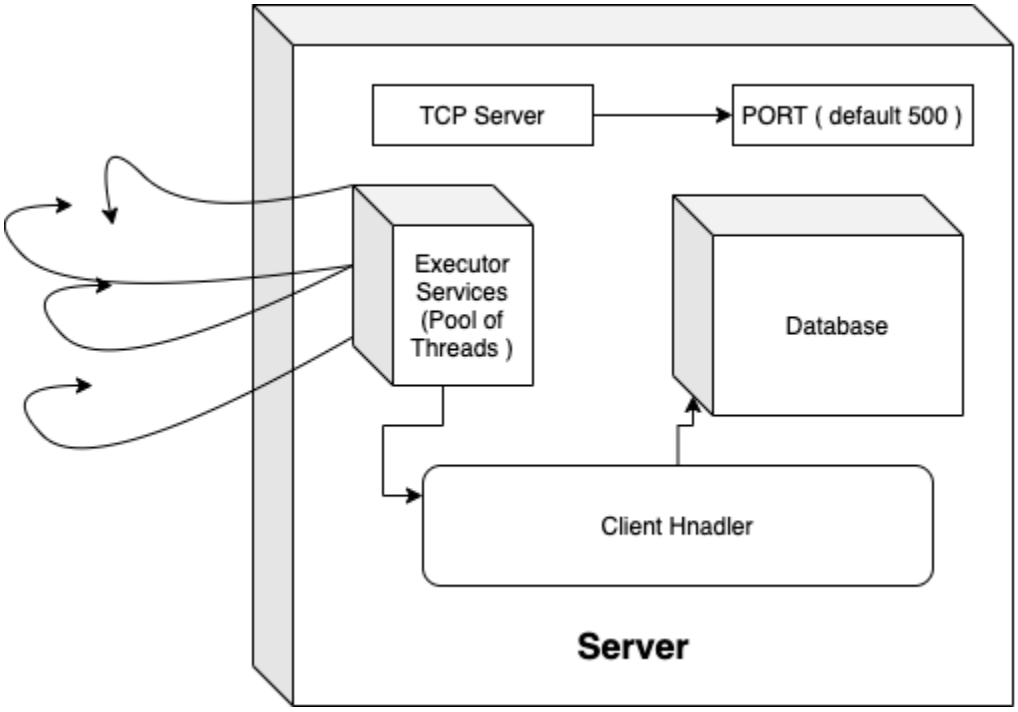
Implement a TCP-based key-value store. The server implements the key-value store and clients make use of it. The server must accept clients' connections and serve their requests for 'get' and 'put' key value pairs. All key-value pairs should be stored by the server only in memory. Keys and values are strings. The client accepts a variable no of command line arguments where the first argument is the server hostname followed by port no. It should be followed by any sequence of "get <key>" and/or "put <key> <value>".

./client 192.168.124.5 5555 put city Kolkata put country India get country get city get Institute India Kolkata

<blank> The server should be running on a TCP port. The server should support multiple clients and maintain their key-value stores separately. Implement authorization so that

only few clients having the role “manager” can access other’s key-value stores. A user is assigned the “guest” role by default. The server can upgrade a “guest” user to a “manager” user

Server Architecture Schema:



For this assignment the language of choice is **Python.**

Architecture:

Server is running via a TCP socket connection on a specified port(in my case it’s 9090) at localhost and has a pool of threads which work by deploying a client handler thread for every incoming client request. For storing the data, I am maintaining two dictionaries named authDict and keyStore.

- authDict keeps the username as key and password as it’s value.

● keyStore is basically a dictionary of dictionary, for each user an inner dictionary is created where the key-value pair is maintained. This is actually working as an in memory database.

Server:

The server consists of two simple functions, namely **handleClient** and **run.** HandleClient function deployes a new thread every time a new user binds with server. Thus maintaining concurrency.

client.py server.py

server.py

```python
import socket
from threading import Thread
import time
import sys
from termcolor import colored

class Server:
    def __init__(self):
        self.serverHost     = '127.0.0.1'
        self.serverPort     = '9090'
        self.socket         = socket.socket(socket.AF_INET, socket.SOCK
        self.authDict       = dict()
        self.keyStore       = dict()
        self.managePassword = 'iammanager'


    def handleClient(self, conn, addr, userName):…




    def run(self):…

if __name__ == "__main__":
    server = Server()
    server.run()
```

Client

Whereas the client side consist of only 1 main function namely run which parses user input sanitize it and perform some queries over server and fetch data from it.

```
client.py > ...
import socket
import time
import sys
from termcolor import colored



class Client():
    def __init__(self):
        self.serverHost = '127.0.0.1' #localhost
        self.serverPort = '9090'
        self.socket     = socket.socket(socket.AF_INET, socket.SOCK_STREAM)



    def run(self):…



if __name__ == "__main__":
    # print(colored("Hello", "red"))
    # print(colored('World','cyan'))
    c = Client()
    c.run()
```

The run method checks user input in a while loop and based on the input given by the user it performs some specific query over server fetch data and display them. The heart of the run function as shown below.

```python
            # parsing happens here
            counter = 0

            # if the function is GET...we need 2 attributes in total
            # i.e GET city_name
            while length > counter:
                if(userType == 'g' and userInput[counter].lower() == 'get'):…

                elif(userType == 'g' and userInput[counter].lower() == 'put'):…

                elif(userType == 'g' and userInput[counter].lower() == 'upgrademe'):…

                elif(userType == 'm' and userInput[counter].lower() == 'put'):…

                elif(userType == 'm' and userInput[counter].lower() == 'get'):…

                else:
                    print("INVALID COMMAND FOUND AT POSITION " + str(counter+1) + "!!")
                    break
```

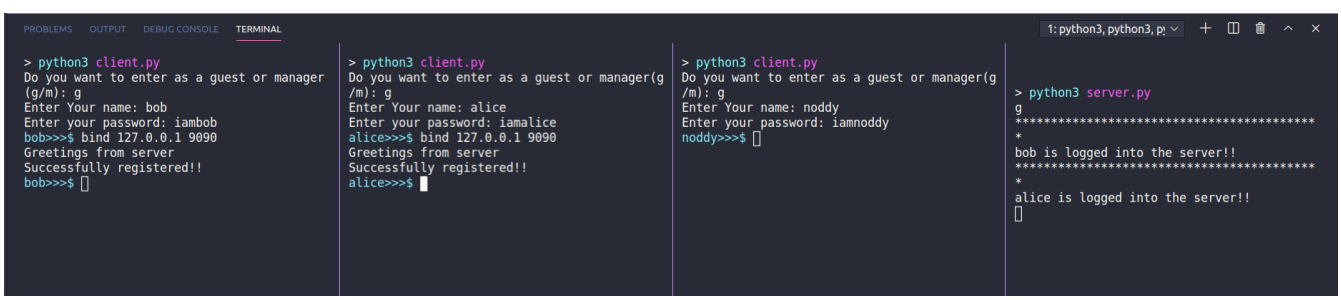
The queries

- Every user need to perform a **bind** operation that will bind the client to the server over a specified port
- By default every user is a guest user. Guest user can **get** and **put** their own data and can perform **upgrade** command to escalate their privilege to a **manager user.**
- Manager user can access and modify data of any user registered in the in memory database.

Output

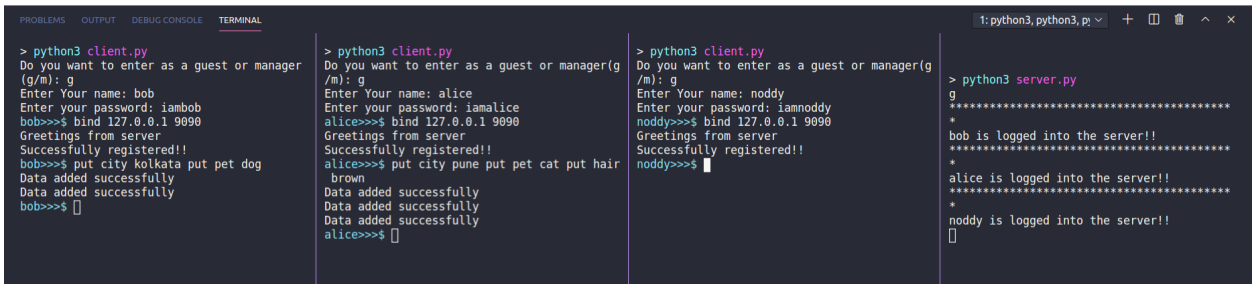
initialization and Connection

Clients and server are getting connected. By default every user is a guest (blue color)

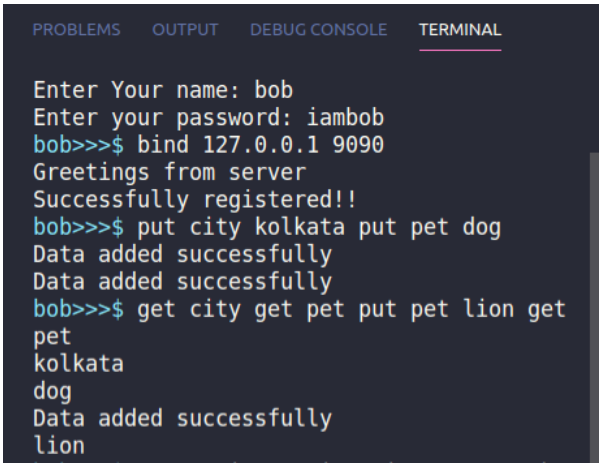
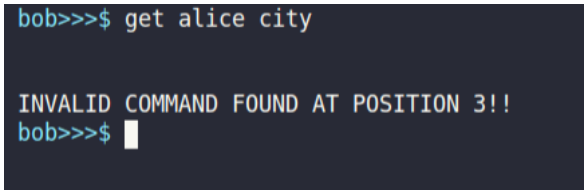


Adding And Retrieving data (access control):

Now we are adding data from the user terminals. We can easily use multiple commands in a single line



Now we will try to retrieve data. We can see in the next diagram that one guest can access its own data and modify it easily. but we cannot get others data.

Access Granted	Access Denied
	

Manager Privilege

Now we can escalate noddy's privilege to the manager role (red instead of blue). Manger can access and modify anyone's record.

```
> python3 client.py
Do you want to enter as a guest or manager(
g/m): g
Enter Your name: noddy
Enter your password: iamnoddy
noddy>>>$ bind 127.0.0.1 9090
Greetings from server
You are now logged into the server!!
noddy>>>$ upgrademe
Enter your manager password: iammanager
noddy>>>#
```

```
> python3 server.py
g
*****************************************
*
bob is logged into the server!!
*****************************************
*
alice is logged into the server!!
*****************************************
*
noddy is logged into the server!!
*****************************************
*
bob is logged into the server!!
*****************************************
*
noddy is logged into the server!!
noddy has now manager privillage!!
```

Manager Access

Now noddy can access anyones data.

```
> python3 client.py
Do you want to enter as a guest or manager(
g/m): g
Enter Your name: noddy
Enter your password: iamnoddy
noddy>>>$ bind 127.0.0.1 9090
Greetings from server
You are now logged into the server!!
noddy>>>$ upgrademe
Enter your manager password: iammanager
noddy>>># get bob city get alice pet
kolkata
cat
noddy>>># put bob city delhi
Data added successfully
noddy>>>#
```

And as we can see, noddy can access anyones data easily. The updation of element also works as bob's city information is now changed to delhi. (it was kolkata before)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

> python3 client.py
Do you want to enter as a guest or manager
(g/m): g
Enter Your name: bob
Enter your password: iambob
bob>>>$ bind 127.0.0.1 9090
Greetings from server
You are now logged into the server!!
bob>>>$ get pet
lion
bob>>>$ get city
kolkata
bob>>>$ get alice city


INVALID COMMAND FOUND AT POSITION 3!!
bob>>>$ get city
delhi
bob>>>$
```

Conclusion

After implementing the required features for a K-V store via TCP socket based server-client model , the concepts such as SERVER-CLIENT models , multithreading , synchronization , concurrency and DSA handling got cleared well. A better query processing mechanism could be implemented via considering all the error factors and informing user for the case of multiple query in a line as stated in the assignment.