INTERNET TECHNOLOGY ASSIGNMENT 1 & 2

Name – Soulib Ghosh
Roll – 0016105010 47
Class – BCSE – IV (A2)
Date – 11/09/19
Sub - Assignment 1 & 2

Assignment 1:

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>".

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.

Assignment 2:

Implement a key-value store using Web socket. 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 as in Assignment 1.

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.

Submit a report on the comparative analysis of the two assignments especially when both roles of manager and guests are considered

1. Introduction:

A brief introduction is given on TCP, web socket, client server model and get-put request respectively.

TCP Socket:

TCP/IP is a reliable protocol used in transport layer of the layered OSI model. It is responsible for process to process communication. However an IP address alone is not sufficient for running network applications, as a computer can run multiple applications and/or services. Just as the IP address identifies the computer, the network port identifies the application or service running on the computer. The combination of IP address and port address is known as socket. That's why we require both IP address and port number for creating a TCP connection. A diagram of the TCP socket is shown in figure 1.

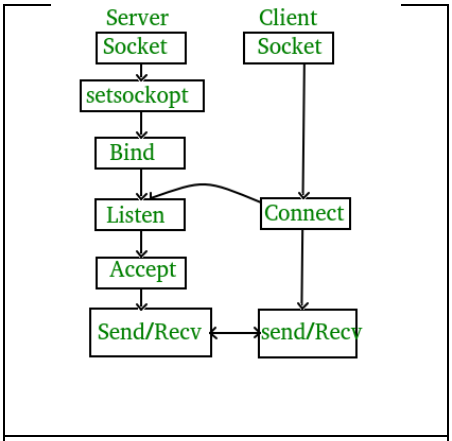

Figure 1: Diagram of a TCP socket

Web Socket:

Web Socket is an advanced technology that allows real-time interactive communication between the client browser and a server. It uses a completely different protocol that allows bidirectional data flow. Web Sockets don't need to send a request in order to respond. It allow full duplex data flow so it just have to listen for any data. With the help of it, one can just listen to the server and it will send you a message when it's available. Due to the absence of thread, programmer don't need to think of the thread synchronization overhead. Web socket is very helpful in cases of real time application and chat app. A schematic diagram of web socket is given in figure 2.

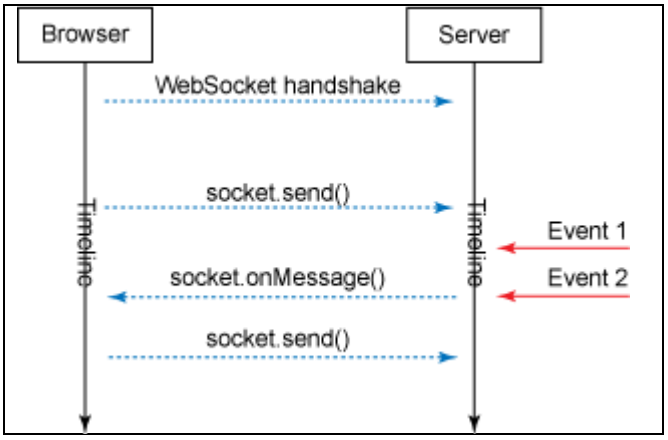

Figure 2: Pictorial representation of a web socket

Client Server Model:

On the other hand, client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web. A pictorial representation of a client server model is shown in figure 3.

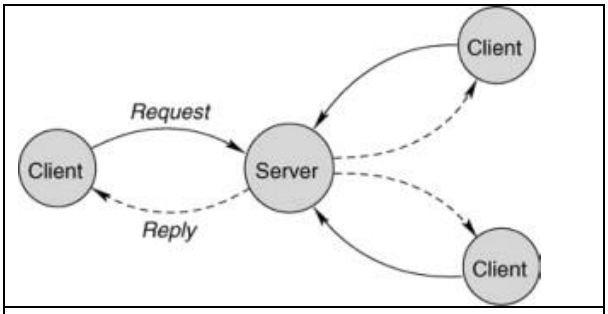

Figure 3: Representation of client server model

Put-get Request:

A key-value pair is a set of two linked data items: a key, which is a unique identifier for some item of data, and the value, which is either the data that is identified or a pointer to the location of that data. Key-value pairs are frequently used in lookup tables, hash tables and configuration files. Every server store the data in the key-value format. Put is a command request given by a client which is followed by two argument - key and a value. Server process a put request from a client and put the key value pair in appropriate place. Get is also a command request from client which is followed by one argument – key. The server operates accordingly and return the value corresponds to that key to the proper client.

2. Proposed Methodology:

The proposed method is divided into three subsection. The initial stage consist of connection establishment. The next stage comprises of request processing. The final stage deals with the response from the server.

Connection establishment:

The proposed approach for both the assignments are quiet similar in terms of designing phase. Initially, TCP socket is created and the connection between client and server is established. On the other hand, the client server is connection is created using a web socket. In TCP socket there is additional overhead of thread synchronization which is done by synchronization mechanism available in PYTHON. Till now a chat environment is created between client and server for both socket.

Request Processing:

In this stage, the problem can be viewed as – a client is sending some string to the server. The server does some processing and again response to the client. When a string arrives to the server from the client, the entire string is split into an array string using space as delimiter. The server side maintains a separate dictionary for each client. During execution server creates a new dictionary each time once a new client arrives. When a put command is encountered, the server takes the next two substring as argument in the order key and value respectively. Then, the server inserts the value in proper place using the key in the dictionary. If an 'upgrade' command arrives, the server gives access to all dictionaries to that client in order to upgrade its status to manager. This process is completely same for the both TCP and web socket.

Response:

Similarly if the server encounters a get command, it take the next sub string as the argument. The server pick out the dictionary of that particular client. It finds the key value in the dictionary. If there is no key then returns an error message else returns the value to the client. This process is also similar for both sockets.

3. Experimental Section:

Both the experiments are carried out using PYTHON. The code snippet is given below for both TCP and web socket separately.

Code for TCP Socket:

Client Side

```
1.  import socket
2.
3.  cmds = ['get','put','upgrade','q']
4.  num_cmd = 3
5.
6.  addr = str(input('Enter ip address : '))
7.  port = int(input('Enter port address : '))
8.
9.  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10. s.connect((addr,port))
11.
12.
13. while True:
14.     user = str(input('Enter user name : '))
15.     s.send(user.encode('utf-8'))
16.     ans = s.recv(1024).decode()
17.     if ans == 'success':
18.         break
19.     else:
20.         print(ans)
21.
22. while True:
23.     x = str(input())
24.     words = x.split()
25.     for i in range(len(words)):
26.         if words[i] in cmds:
27.             x=words[i]
28.             if x == 'upgrade' or x=='q':
29.                 s.send(x.encode('utf-8'))
30.                 x2 = s.recv(1024).decode()
```

```
                print(x2)
                if x=='q':
                    break
        elif (i==len(words)-1) or (words[i+1] in cmds):
            x=x+' '+words[i]
            boo=False
            for j in range(num_cmd):
                boo = boo or x.startswith(cmds[j])
            if boo:
                s.send(x.encode('utf-8'))
                x2 = s.recv(1024).decode()
                print(x2)
            else:
                print('wrong message format')
        else:
            x=x+' '+words[i]
    if x=='q':
        break
```

Server Side:

```
import socket
from _thread import *
import threading

srvr_lock = threading.Lock()

class Client:
    def __init__(self, a, c):
        self.name = a
        self.conn = c
        self.dct = {'role':'guest'}

clients = {}

def operate(c):
    while True:
        data = c.conn.recv(1024).decode()
        arr = data.split()

        if arr[0] == 'q':
            c.conn.send('exiting'.encode('utf-8'))
            break

        elif arr[0] == 'put':
            if len(arr) == 3:
                c.dct[arr[1]] = arr[2]
                c.conn.send('done'.encode('utf-8'))
            else:
                c.conn.send('wrong message format'.encode('utf-8'))

        elif arr[0] == 'get':
            if len(arr) == 2:
                if arr[1] in c.dct:
                    c.conn.send(c.dct[arr[1]].encode('utf-8'))
                else:
                    c.conn.send('key not found'.encode('utf-8'))

            elif len(arr) == 3:
                if c.dct['role'] == 'guest':
```

```
                    c.conn.send('you are not allowed to access keys of other users'.enc
ode('utf-8'))
                elif arr[1] in clients:
                    c2 = clients[arr[1]]
                    if arr[2] in c2.dct:
                        c.conn.send(c2.dct[arr[2]].encode('utf-8'))
                    else:
                        c.conn.send('key not found'.encode('utf-8'))
                else:
                    c.conn.send('user not found'.encode('utf-8'))

            else:
                c.conn.send('wrong message format'.encode('utf-8'))

        elif arr[0] == 'upgrade':
            if c.dct['role'] == 'guest':
                c.dct['role'] = 'manager'
                c.conn.send('upgraded to manager'.encode('utf-8'))
            else:
                c.conn.send('already a manager'.encode('utf-8'))

        else:
            c.conn.send('wrong message format'.encode('utf-8'))

    c.conn.close()
    del clients[c.name]


def main():
    addr = "127.0.0.1"
    port = 1601
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((addr,port))
    s.listen(5)

    while True:
        c, addr2 = s.accept()
        while True:
            x = c.recv(1024).decode()
            if x in clients:
                c.send('duplicate user name'.encode('utf-8'))
            else:
                c.send('success'.encode('utf-8'))
                print('Connected to '+x+':'+str(addr2))
                break
        cc = Client(x,c)
        clients[x] = cc
        # srvr_lock.acquire()
        start_new_thread(operate, (cc,))

    s.close()

if __name__ == '__main__':
    main()
```

Output:

Some user given input and their corresponding output for TCP socket are shown in figure 4. In this scenario, three client are considered which work simultaneously. Additionally it is also shown when a user is upgraded to manager.

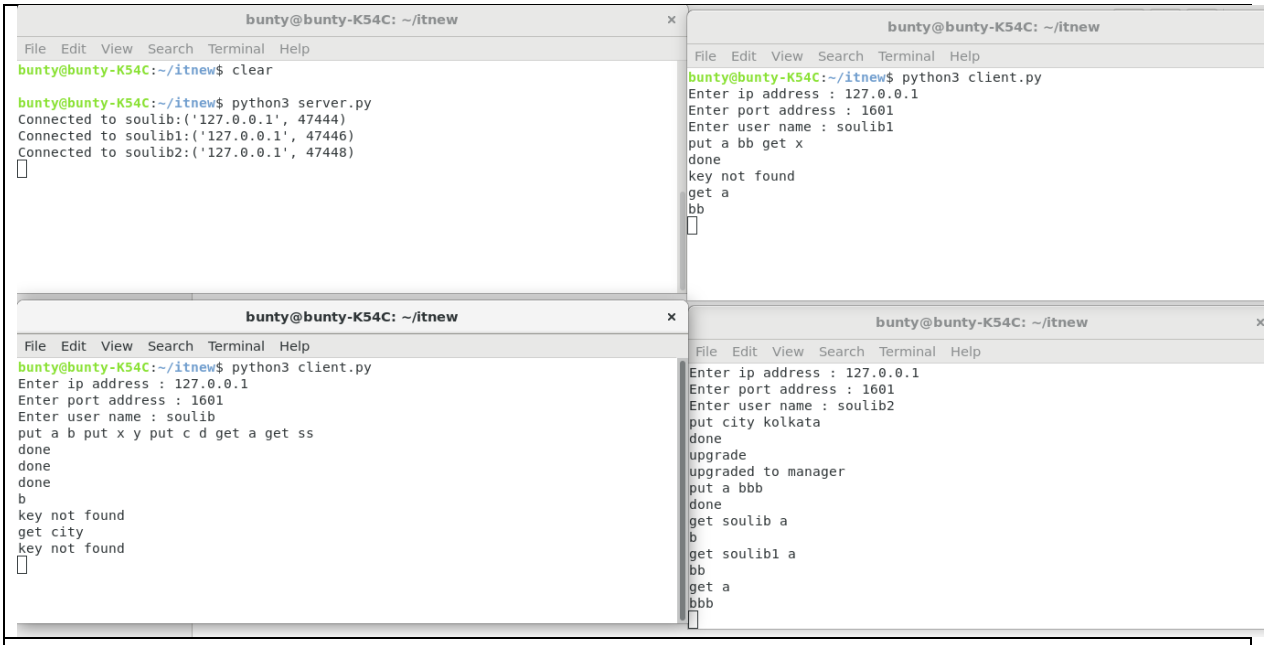

Figure 4: Output of TCP socket.

Code for Web Socket:

Client Side:

```
1.  import asyncio
2.  import websockets
3.
4.  addr = str(input('>>Enter ip address : '))
5.  port = int(input('>>Enter port address : '))
6.
7.  cmds = ['get','put','upgrade','q']
8.  num_cmd = 4
9.
10. async def hello():
11.     uri = "ws://"+addr+":"+str(port)
12.     async with websockets.connect(uri) as websocket:
13.
14.         while True:
15.             if not websocket.open:
16.                 print("Websocket not open. Trying to reconnect")
17.                 websocket = await websockets.connect(uri)
18.             user = str(input('>>Enter user name : '))
19.             await websocket.send(user)
20.             ans = await websocket.recv()
21.             if ans == 'success':
22.                 print('user created')
23.                 break
24.             else:
25.                 print(str(ans))
26.
27.         while True:
28.             if not websocket.open:
29.                 print("Websocket not open. Trying to reconnect")
30.                 websocket = await websockets.connect(uri)
31.             x = str(input(">>"))
```

```
            words = x.split()
            for i in range(len(words)):
                if words[i] in cmds:
                    x=words[i]
                    if x == 'upgrade' or x=='q':
                        await websocket.send(x)
                        x2 = await websocket.recv()
                        print(str(x2))
                        if x=='q':
                            break
                elif (i==len(words)-1) or (words[i+1] in cmds):
                    x=x+' '+words[i]
                    boo=False
                    for j in range(num_cmd):
                        boo = boo or x.startswith(cmds[j])
                    if boo:
                        await websocket.send(x)
                        x2 = await websocket.recv()
                        print(str(x2))
                    else:
                        print('wrong message format')
                else:
                    x=x+' '+words[i]
            if x=='q':
                break

asyncio.get_event_loop().run_until_complete(hello())
```

Server Side:

```
import asyncio
import websockets

class Client:
    def __init__(self, a):
        self.name = a
        self.dct = {'role':'guest'}

clients = {}

async def hello(websocket, path):
    while True:
        x = await websocket.recv()
        if x in clients:
            await websocket.send('duplicate user name')
        else:
            await websocket.send('success')
            print('Connected to '+x)
            break
    c = Client(x)
    clients[x] = c

    # async for data in websocket:
    while True:
        data = await websocket.recv()
        arr = data.split()

        if arr[0] == 'q':
            await websocket.send('exiting')
            break

```

```
        elif arr[0] == 'put':
            if len(arr) == 3:
                c.dct[arr[1]] = arr[2]
                await websocket.send('done')
            else:
                await websocket.send('wrong message format')

        elif arr[0] == 'get':
            if len(arr) == 2:
                if arr[1] in c.dct:
                    await websocket.send(c.dct[arr[1]])
                else:
                    await websocket.send('key not found')

            elif len(arr) == 3:
                if c.dct['role'] == 'guest':
                    await websocket.send('you are not allowed to access keys of other u
sers'.encode('utf-8'))
                elif arr[1] in clients:
                    c2 = clients[arr[1]]
                    if arr[2] in c2.dct:
                        await websocket.send(c2.dct[arr[2]])
                    else:
                        await websocket.send('key not found')
                else:
                    await websocket.send('user not found')

            else:
                await websocket.send('wrong message format')

        elif arr[0] == 'upgrade':
            if c.dct['role'] == 'guest':
                c.dct['role'] = 'manager'
                await websocket.send('upgraded to manager')
            else:
                await websocket.send('already a manager')

        else:
            await websocket.send('wrong message format')

    del clients[c.name]

addr = ""
port = 1603
start_server = websockets.server.serve(hello, addr, port, ping_interval = 1000, ping_ti
meout = 1000)
asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

Output:

Some user given input and their corresponding output for web socket are shown in figure 5. In this scenario, three client are considered which work simultaneously. Additionally it is also shown when a user is upgraded to manager.

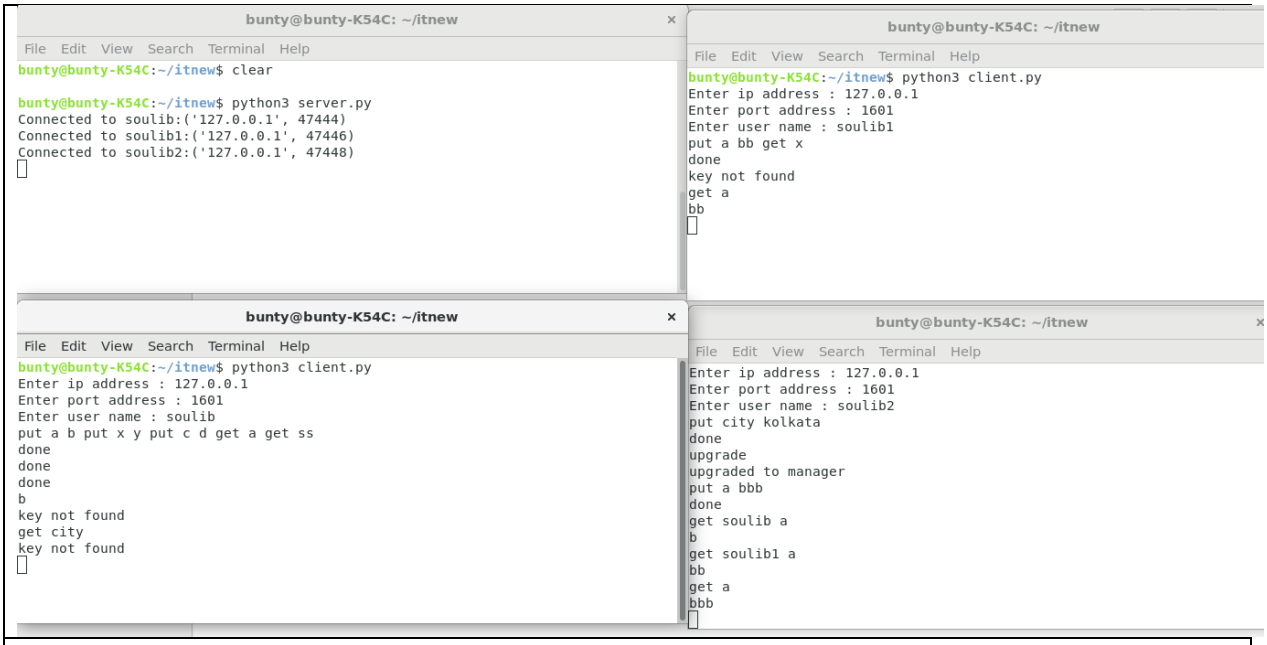

Figure 5: Output of web socket

4. Conclusion:

Web Sockets enable the server and client to send messages to each other at any time, after a connection is established, without an explicit request by one or the other. In a challenge-response system there is no way for clients to know when new data is available for, with Web sockets the server can push new data at any time which makes them the better candidate for “real-time” applications.
