INTERNET TECHNOLOGIES LAB REPORT

NAME: ANURAN CHAKRABORTY
ROLL NO.: 20
CLASS: BCSE-III
SECTION: A1

ASSIGNMENT NUMBER: 1

PROBLEM STATEMENT:

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.

CODE:

The assignment has been implemented in python3.6.

common.py stores the commonly used functions by server and client

```
import socket

portServer=12345

# Function to create a socket and bind it to a port
def createSocket(port):
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind(('', port))
        return s

# Function to receive a connection
def allowConn(s):
        s.listen(5)
        c, addr=s.accept()
        return c, addr

# Function to create a socket and connect to it
def createConn(port,ip=''):
        sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((ip,port))
        return sock

# Function to send a frame
def send_frame(frame, c):
        # Send the frame to the other process
        c.send(frame.encode())
```

client.py contains the client-side code

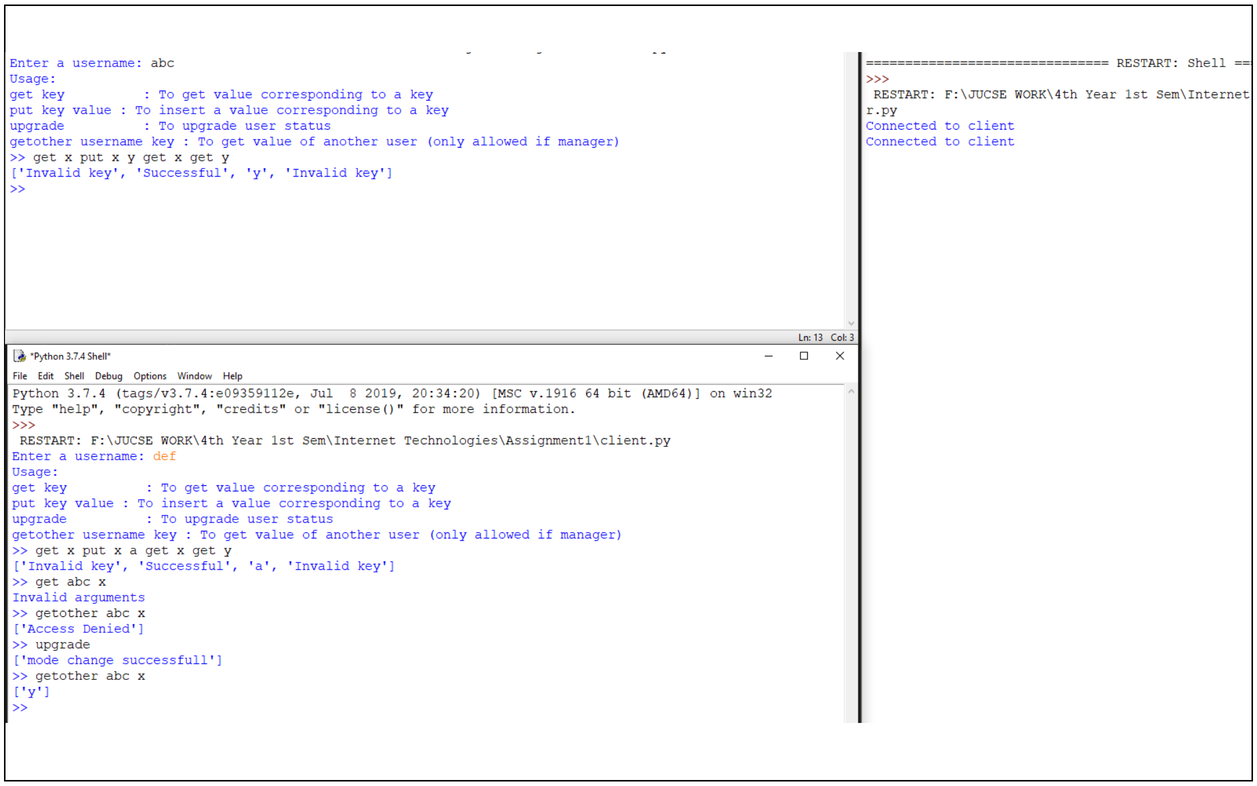
```
 import socket
import threading
import common as co
import pickle
import sys

# Function to return a dictionary based on the request
def parseArgs(args):
        req=[]
        i=0
        while i<(len(args)):

                if(args[i].lower()=='get'):
                        if(i==len(args)-1 or args[i+1].lower()=='put'): # Error
case
                                return 0,req
                        else:
                                req.append({'method':'get','key':args[i+1]})
                                i=i+1

                elif(args[i].lower()=='put'):
                        if(i==len(args)-2): # Error case
                                return 0,req
                        else:

        req.append({'method':'put','key':args[i+1],'value':args[i+2]})
                                i=i+2

                elif(args[i].lower()=='getother'):
                        if(i==len(args)-2): # Error case
                                return 0,req
                        else:

        req.append({'method':'getother','key':args[i+2],'username':args[i+1]})
                                i=i+2

                elif(args[i].lower()=='upgrade'):
                        req.append({'method':'upgrade'})
                else:
                        return 0,req
                i=i+1

        return 1,req

sockClient=co.createConn(port=int(sys.argv[2]),ip=sys.argv[1])

uname=input('Enter a username: ')
sockClient.sendall(uname.encode())

print('Usage:')
print('get key          : To get value corresponding to a key')
print('put key value : To insert a value corresponding to a key')
print('upgrade          : To upgrade user status')
print('getother username key : To get value of another user (only allowed if
manager)')

while(True):
        # Take input
        request=input('>> ')
        if(request.lower()=='exit'):
                break
        retVal,req=parseArgs(request.split(' '))

        if(retVal==0):
                print('Invalid arguments')
                continue

        # print(req)

        req=pickle.dumps(req)
        # Send the dictionary through socket
        sockClient.sendall(req)

        # Wait for response
        response=sockClient.recv(1024)
        response=pickle.loads(response)

        print(response)
```

server.py contains the server-side code.

```
import socket
import threading
import common as co
import pickle

# Class to store key value for each client
class KeyValueClient:

        def __init__(self,username):
                self.valstore={}
                self.mode='guest'
                self.username=username

        def _change_mode(self):
                self.mode='admin'
        def _getValue(self,key):
                if(key not in self.valstore):
                        return 'Invalid key'
                return self.valstore[key]

        def _putValue(self,key,value):

                self.valstore[key]=value
                return 'Successful'

        # Function to take action on the requests
        def takeAction(self,req):

                res=[]

                for reqs in req:
                        if(reqs['method'].lower()=='get'):
                                res.append(self._getValue(reqs['key']))

                        elif(reqs['method'].lower()=='put'):

        res.append(self._putValue(reqs['key'],reqs['value']))

                        elif(reqs['method'].lower()=='upgrade'):
                                self._change_mode()
                                res.append('mode change successfull')

                        elif(reqs['method'].lower()=='getother'):
                                if(self.mode=='guest' and
self.username!=reqs['username']):
                                        res.append('Access Denied')
                                elif(self.username==reqs['username'] or
self.mode=='admin'):

                                        if(reqs['username'] in global_dict):

        res.append(global_dict[reqs['username']]._getValue(reqs['key']))
                                        else:
                                                res.append('Invalid username')

                return res

sockServer=co.createSocket(co.portServer)
global_dict={}
```

```
# Function to service a client
def serviceClient(client, clientAddr):

        while True:

                requestC=clientAddr.recv(1024) # Receive the request 
dictionary
                requestC=pickle.loads(requestC)
                res=client.takeAction(requestC)
                res=pickle.dumps(res)
                clientAddr.sendall(res)

def allow_new_conn():

        while(True):
                # Wait for a connection
                sockServer.listen(10)
                cAddr, addrServer=sockServer.accept()
                print('Connected to client')
                # Fetch username
                uname=cAddr.recv(1024).decode()
                client=KeyValueClient(uname) # Create client by that username
                global_dict[uname]=client

                # Start a new thread for the sender
                sendThread=threading.Thread(target=serviceClient, 
args=[client,cAddr])
                sendThread.start()

allow_new_conn()
```

OUTPUT:



```
Enter a username: abc
Usage:
get key          : To get value corresponding to a key
put key value : To insert a value corresponding to a key
upgrade          : To upgrade user status
getother username key : To get value of another user (only allowed if manager)
>> get x put x y get x get y
['Invalid key', 'Successful', 'y', 'Invalid key']
>>
```

```
============================== RESTART: Shell ==
>>>
 RESTART: F:\JUCSE WORK\4th Year 1st Sem\Internet
r.py
Connected to client
Connected to client
```

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: F:\JUCSE WORK\4th Year 1st Sem\Internet Technologies\Assignment1\client.py
Enter a username: def
Usage:
get key          : To get value corresponding to a key
put key value : To insert a value corresponding to a key
upgrade          : To upgrade user status
getother username key : To get value of another user (only allowed if manager)
>> get x put x a get x get y
['Invalid key', 'Successful', 'a', 'Invalid key']
>> get abc x
Invalid arguments
>> getother abc x
['Access Denied']
>> upgrade
['mode change successfull']
>> getother abc x
['y']
>>
```

On the left two clients are started and on the right the server. The program can handle multiple commands in a single line. The client registers with a username. Then client ‘abc’ asks for the key x. It is not present and hence the output ‘Invalid key’. Client ‘def’ asks for the key of ‘abc’ but at the beginning it is a guest so ‘Access Denied’. Later after ‘upgrade’ it can view.

ASSIGNMENT NUMBER: 2

PROBLEM STATEMENT:

Implement a key-value store using Websocket. 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.

CODE:

The assignment has been implemented in python3.6.

client.py contains the client-side code

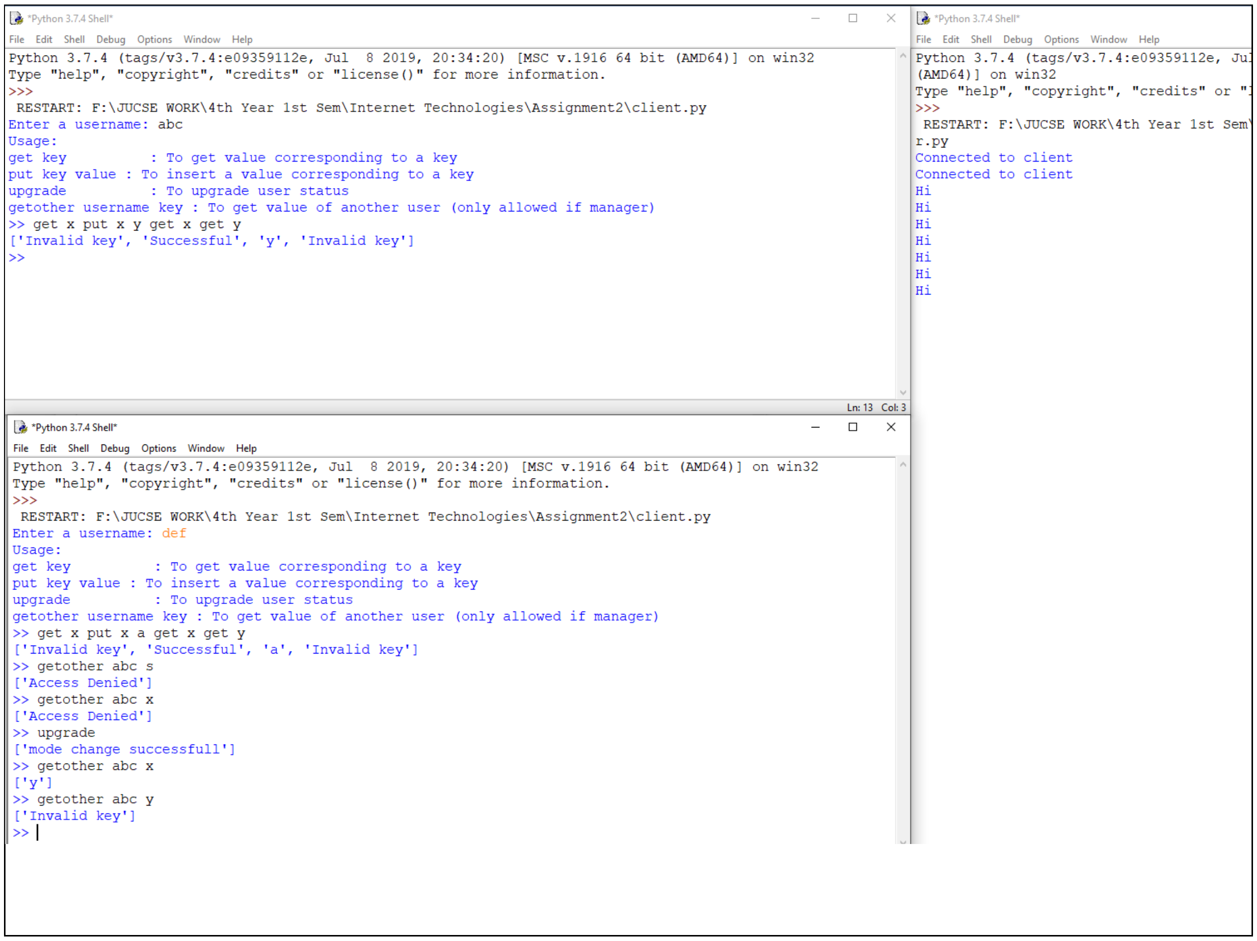
```
import asyncio
import websockets
import pickle
import sys

# Function to return a dictionary based on the request
def parseArgs(args):

        req=[]
        i=0
        while i<(len(args)):

                if(args[i].lower()=='get'):
                        if(i==len(args)-1 or args[i+1].lower()=='put'): # Error
case
                                return 0,req
                        else:
                                req.append({'method':'get','key':args[i+1]})
                                i=i+1

                elif(args[i].lower()=='put'):
                        if(i==len(args)-2): # Error case
                                return 0,req
                        else:

        req.append({'method':'put','key':args[i+1],'value':args[i+2]})
                                i=i+2

                elif(args[i].lower()=='getother'):
                        if(i==len(args)-2): # Error case
                                return 0,req
                        else:

        req.append({'method':'getother','key':args[i+2],'username':args[i+1]})
                                i=i+2

                elif(args[i].lower()=='upgrade'):
                        req.append({'method':'upgrade'})
                else:
                        return 0,req
                i=i+1

        return 1,req


async def clientRun():
        ws_url='ws://'+sys.argv[1]+':'+str(sys.argv[2])
        # async with websockets.connect(ws_url) as sockClient:
        sockClient=await websockets.connect(ws_url)
        # Accept username
        uname=input('Enter a username: ')
        await sockClient.send(uname)

        print('Usage:')
        print('get key          : To get value corresponding to a key')
        print('put key value : To insert a value corresponding to a key')
        print('upgrade          : To upgrade user status')
        print('getother username key : To get value of another user (only
allowed if manager)')

        while(True):
                # Take input
                request=input('>> ')
                if(request.lower()=='exit'):
                        break
                retVal,req=parseArgs(request.split(' '))

                if(retVal==0):
                        print('Invalid arguments')
                        continue

                # print(req)

                req=pickle.dumps(req)
                # Send the dictionary through socket
                await sockClient.send(req)

                # Wait for response
                try:
                        response=await sockClient.recv()
                except:
                        # Reconnect
                        print('Reconnecting...')
                        sockClient=await websockets.connect(ws_url)
                        # response=await sockClient.recv()

                response=pickle.loads(response)

                print(response)

asyncio.get_event_loop().run_until_complete(clientRun())
```

server.py contains the server-side code.

```
import asyncio
import websockets

import socket
import threading
import pickle

# Class to store key value for each client
class KeyValueClient:

        def __init__(self,username):
                self.valstore={}
                self.mode='guest'
                self.username=username

        def _change_mode(self):
                self.mode='admin'

        def _getValue(self,key):
                if(key not in self.valstore):
                        return 'Invalid key'
                return self.valstore[key]

        def _putValue(self,key,value):

                self.valstore[key]=value
                return 'Successful'

        # Function to take action on the requests
        def takeAction(self,req):

                res=[]

                for reqs in req:
                        if(reqs['method'].lower()=='get'):
                                res.append(self._getValue(reqs['key']))

                        elif(reqs['method'].lower()=='put'):

        res.append(self._putValue(reqs['key'],reqs['value']))

                        elif(reqs['method'].lower()=='upgrade'):
                                self._change_mode()
                                res.append('mode change successfull')

                        elif(reqs['method'].lower()=='getother'):
                                if(self.mode=='guest' and
self.username!=reqs['username']):
                                        res.append('Access Denied')
                                elif(self.username==reqs['username'] or
self.mode=='admin'):

                                        if(reqs['username'] in global_dict):

        res.append(global_dict[reqs['username']]._getValue(reqs['key']))
                                        else:
                                                res.append('Invalid username')

                return res

global_dict={}

# Function to service a client
async def serviceClient(clientAddr,path):

        print('Connected to client')
        # Fetch username
        uname=await clientAddr.recv()
        client=KeyValueClient(uname) # Create client by that username
        global_dict[uname]=client

        while True:

                requestC=await clientAddr.recv() # Receive the request
dictionary
                requestC=pickle.loads(requestC)
                res=client.takeAction(requestC)
                res=pickle.dumps(res)
                print('Hi')
                await clientAddr.send(res)
```

```
start_server = websockets.server.serve(serviceClient, '', 8765,
ping_timeout=100000, ping_interval=100000)
asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

OUTPUT:



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: F:\JUCSE WORK\4th Year 1st Sem\Internet Technologies\Assignment2\client.py
Enter a username: abc
Usage:
get key          : To get value corresponding to a key
put key value : To insert a value corresponding to a key
upgrade          : To upgrade user status
getother username key : To get value of another user (only allowed if manager)
>> get x put x y get x get y
['Invalid key', 'Successful', 'y', 'Invalid key']
>>
```

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Ju
(AMD64)] on win32
Type "help", "copyright", "credits" or "
>>>
 RESTART: F:\JUCSE WORK\4th Year 1st Sem\
r.py
Connected to client
Connected to client
Hi
Hi
Hi
Hi
Hi
Hi
Hi
```

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: F:\JUCSE WORK\4th Year 1st Sem\Internet Technologies\Assignment2\client.py
Enter a username: def
Usage:
get key          : To get value corresponding to a key
put key value : To insert a value corresponding to a key
upgrade          : To upgrade user status
getother username key : To get value of another user (only allowed if manager)
>> get x put x a get x get y
['Invalid key', 'Successful', 'a', 'Invalid key']
>> getother abc s
['Access Denied']
>> getother abc x
['Access Denied']
>> upgrade
['mode change successfull']
>> getother abc x
['y']
>> getother abc y
['Invalid key']
>> |
```

On the left two clients are started and on the right the server. The program can handle multiple commands in a single line. The client registers with a username. Then client ‘abc’ asks for the key x. It is not present and hence the output ‘Invalid key’. Client ‘def’ asks for the key of ‘abc’ but at the beginning it is a guest so ‘Access Denied’. Later after ‘upgrade’ it can view.

COMPARATIVE ANALYSIS:

TCP Socket	Websocket
For a non-blocking TCP socket it will send data if the size of the data is less than the buffer size. If it is blocking it will wait for the buffer to be full and then send the data. Larger data may be fragmented and transmitted	Websocket can only send data if the data size is less than the buffer size. Websockets do not fragment data.
TCP sockets are half duplex i.e. while it receives data from a host it cannot simultaneously send data to the host.	Websockets are full duplex connections which allow simultaneous sending and receiving of data
In terms of the coding for interaction with multiple clients using TCP sockets threads needs to be manually created.	Threads need not be manually created and are handled by the library.