COMP3331 Assignment

Luoyou Zhao z5225024

The python version I used in this assignment is 3.7.4, I am going to introduce the code in three part.

The commands is different with the example Interactions shown on Webcms. After loging in the client, please type "help" for more information.

Message and ContactLog

The Message.py and contactLog.py contains several classes and function I used in client and server.

Message class is used in information transmission. The following two functions help change the format of message to the requirement of tcp and udp.

```
class Message:
    def __init__(self, command, data):
        self.command = command
        self.data = data

def packMessage(command, data):
    mes = Message(command, data)
    return pickle.dumps(mes)

def unpackMessage(mes):
    return pickle.loads(mes)
```

User class stores username and password of the user, and it can also check if they are valid.

```
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password
        self.wrongTime = 0
        self.block = False

    def check(self, username, password):
        return self.username == username and self.password == password
```

Log class stores tempID, its start time and its expire time.

```
class Log:
    def __init__(self, temp_id, start_time, expire_time):
        self.tempId = temp_id
        self.startTime = start_time
        self.expireTime = expire_time

    def __eq__(self, other):
        return self.tempId == other.tempId and self.startTime ==
other.startTime and self.expireTime == other.expireTime

    def __hash__(self):
        return hash((self.tempId, self.startTime, self.expireTime))

    def isValid(self):
        current_time = time.time()
        if txt2time(self.startTime) < current_time < txt2time(self.expireTime):
            return True
        else:
            return False

    def isLaterThan(self, log):
        return txt2time(self.startTime) > txt2time(log.startTime)

def txt2time(string):
    return time.mktime(time.strptime(string, timeFormat))

def time2txt(t):
    return time.strftime(timeFormat, time.localtime(t))

def generateLog(temp_id):
    return Log(temp_id, time2txt(time.time()), time2txt(time.time() + 900))
```

Server Part

First, the server open files credentials.txt and tempIDs.txt, stores the information in two dictionaries accounts{Username: Password} and tempIDs{Log: Username}.

Dictionary i_tempIDs{Username: Log} stores the newest tempID for every user.

After the server making connection with a new client, the server will give this client a unique port number and start a new threading for this client. The main program will wait for the next client.

In the login part, server use user class to record the number of unsuccessful authentication attempts. After three times, it will set user.block to true, and start to new thread to change it back after block_duration secs. Server uses dictionary accounts{} to validate the usename and password.

When server receive download command, it will take tempID from i_tempIDs{}, check if it is valid, or it will generate a new ID and send it to client.

When server receive upload command, it will check the logs in tempIDs{}, finds all correspond username and print them out.

Client Part

The client firstly check username and password input by user. After receiving the validation from server, the client read file z5225024_contactlog.txt, store the logs in contactLogs{Log: time}. The time is current time.

Then, client start two threading, the first threading receive Beacon from other client. If the log is valid, it stores the Beacon in contactLogs{}

The second theading is to update the contactLogs{}, it will go through the whole dictionary and compare its time with current time. If the log has already stored in contactLogs{} for three minutes (which is 180 secs), this log will be deleted. Because the length of contactLogs is changing by receive Beacon(), so the updateBeacon() will sleep for 10 secs after each iteration(And the beacon may not be deleted immediately after three minutes, it will have a maximum 10 secs delay).

Altough I know this is not the best way to update beacons, I could not find a better solution. This may causes potential bugs in some situations (such as sending beacons frequently) that I did not meet when I was testing.

```
def updateBeacon():
    while True:
        for log in list(contactLogs.keys()):
            if time.time() > contactLogs[log] + 180:
                del contactLogs[log]
        for log in list(contactLogs.keys()):
            f = open("z5225024_contactlog.txt", mode='w')
            f.write(log.tempId + " " + log.startTime + " " + log.expireTime +
"\n")
            f.close()
        time.sleep(10)
```