Assignment Report

Content:

1. Language requirement & environment setup
2. Program design
3. Data structure design
4. Application layer message format
5. How system work

Part 01: Language requirement and environment setup

Language	Python
Version	3.8.10
Libraries Requirement	Socket, threading, pickle, datetime, time, sys, os

Part 02: Program design

There are 2 main sections in this assignment design --- Server and Client.

1) Server

The server starts a new ClientThread when there is a user login. In the ClientThread, there defines many self-functions in format "process_*******()", which can process different client command and cases. In the run() of ClientThread, there is a while loop which can continuously receive and send message with the client.

2) Client

The Client has 4 different classes – (PrivateSendThread, PrivateReceiveThread, SendThread, ReceiveThread). The SendThread and ReceiveThread are the main part which process all the user input and server reply. For the p2p section, Client will start up with its own "PrivateSendthread" and "PrivateReceiveThread", and extra socket connected to the target client. This can allow the client to send-receive to all the private tunnels and also to the server.

Part03: Data Structure design

- Client:
  1. Global variables

Variable Name	Type	Description
message_send	Message (defined in Message.py)	save message that need to send via Client socket
message_received	Message (defined in Message.py)	save message that received by Client socket
confirm_wait	Boolean	**True**: when there is requester asking for private messaging. **False**: when there is no private requester
requester_list	List of string	list of private messaging requester
username	String	Client username
password	String	Client password
private_send_thread_list	List of PrivateSendThread (defined in TCPClient3.py)	Send_Threads for all private messaging tunnel
private_receive_thread_list	List of PrivateReceiveThread (defined in TCPClient3.py)	Receive_Threads for all private messaging tunnel

- Server
  1. Global variables

Variable Name	Type	Description
UserData	Dictionary format: {user: passwd, ……}	Load data from *credential.txt,* update when new user login.
OnLine_list	List of tuples format: [(user, login_time), ……]	Data of online_users, update when login or logout
Log_history	List of tuples format: [(user, login_time), ……]	Data of user log history, only update when login.
login_blocked_list	List of tuples format: [(user, TimeThread), ……]	Information of blocked accounts and relevant Timers.
clientThread_list	List of ClientThread	Store all the ClientThreads
blocker_list	Dictionary format: {user: [blocked_users], ……}	The unidirectional relationship of blocking
stored_message_list	Dictionary format: {user: [messages], ……}	All stored messages and their target users

(Note: for the local variables, please check the relevant code, the variable names can explain its role)

Part 04: Application layer message format

```python
class MessageType(Enum):
    NOCOMMAND = -1
    ARGUMENT_ERROR = -2

    LOGIN_USERNAME = 0.1
    LOGIN_PASSWD = 0.2
    LOGIN_NEWPASSWD = 0.3

    MESSAGE = 1
    BROADCAST = 2
    WHOELSE = 3
    WHOELSESINCE = 4
    BLOCK = 5
    UNBLOCK = 6
    LOGOUT = 7

    STARTPRIVATE = 8
    TELL_TARGET_USER_SETUP_PRIVATE_SENDERTHREAD = 8.1
    PRIVATE = 9
    STOPPRIVATE = 10
    STOPPRIVATE_PRIVATE_STOPPED = 10.1

    YES = 11
    NO = 12
```

```python
class ServerMessageType(Enum):
    ERROR = -100

    REQUEST_NEWUSER = 101
    REQUEST_NEEDPASSWORD = 102
    ANNONCEMENT = 103
    TIMEOUT = 104
    ACCOUNT_BLOCK = 105

    ASK_FOR_PRIVATE_CONNECTION = 106
    SEND_REQUESTER_SOCKET_ADDRESS = 107
    SETUP_PRIVATE_SENDERTHREAD = 108

    STOP_AND_DELETE_PRIVATE = 109
    STOPPED_PRIVATE_CONFIRM = 110

    LOGOUT_ANNOUNCEMENT = 111
```

```python
class Message(object):
    def __init__(self, content, message_type):
        self.content = content
        self.type = message_type

    def getContent(self):
        return self.content

    def getType(self):
        return self.type
```

```python
class ServerMessage(object):
    def __init__(self, content, reply_type):
        self.content = content
        self.type = reply_type

    def getContent(self):
        return self.content

    def getType(self):
        return self.type
```

For the Message Types of Clients and Server, They are all defined in the Message.py
As the diagrams shown, there are 19 MessageTypes and 12 ServerMessageTypes. The Message class are the packet that send from Client to Server, and ServerMessage class is the packet that send from Server to Client. Both of them have two self-functions to get the message content and message type.

Part 05: How the system work

Client:

- Non-P2P Command:

In the client, the user input will be processed by the main SendThread, and the server reply will be processed by the main ReceiveThread. All the command and replies will be classified based on its MessageType or ServerMessageType, and then process in different if-statement cases.

```python
class ReceiveThread(Thread):
    def __init__(self, clientSocket:socket): ···

    def run(self):
        global message_received
        global username
        global password
        global confirm_wait

        while self.Alive:
            data = self.clientSocket.recv(1024)
            message_received = pickle.loads(data)

            if message_received.getType() == ServerMessageType.ANNONCEMENT: ···
            elif message_received.getType() == ServerMessageType.LOGOUT_ANNOUNCEMENT: ···
            elif message_received.getType() == ServerMessageType.TIMEOUT: ···
            elif message_received.getType() == ServerMessageType.ERROR: ···
            elif message_received.getType() == ServerMessageType.ASK_FOR_PRIVATE_CONNECTION: ···
            elif message_received.getType() == ServerMessageType.SEND_REQUESTER_SOCKET_ADDRESS:
            elif message_received.getType() == ServerMessageType.SETUP_PRIVATE_SENDERTHREAD: ···
            elif message_received.getType() == ServerMessageType.STOP_AND_DELETE_PRIVATE: ···
            elif message_received.getType() == ServerMessageType.STOPPED_PRIVATE_CONFIRM: ···
```

```python
class SendThread(Thread):
    def __init__(self, clientSocket:socket): ···

    def run(self):
        global message_send
        global message_received
        global username
        global password
        global confirm_wait      You, 2 days ago • generate format and also redesign client

        while self.Alive:

            message_send = input()
            [self.message_content, self.message_type] = MessageContentByType(message_send)

            if confirm_wait == True and requester_list != []: ···
            else:
                if self.message_type == MessageType.NOCOMMAND or self.message_type == MessageType.YES or self.message_type == MessageType.NO: ···
                elif self.message_type == MessageType.MESSAGE: ···
                elif self.message_type == MessageType.BROADCAST: ···
                elif self.message_type == MessageType.WHOELSE: ···
                elif self.message_type == MessageType.WHOELSESINCE: ···
                elif self.message_type == MessageType.BLOCK: ···
                elif self.message_type == MessageType.UNBLOCK: ···
                elif self.message_type == MessageType.LOGOUT: ···
                elif self.message_type == MessageType.STARTPRIVATE: ···
                elif self.message_type == MessageType.PRIVATE: ···
                elif self.message_type == MessageType.STOPPRIVATE: ···
                elif self.message_type == MessageType.ARGUMENT_ERROR: ···
```

- P2P Command:

For the p2p commands, once the SendThread get user input as a p2p command, it will look through the private_thread_lists and process this command with the relevant PrivateThreads or send it to the server to ask target client making private messaging tunnel. The PrivateSendThread take the role of private message sending, and the PrivateReceiveThread take the role of private message receiving.

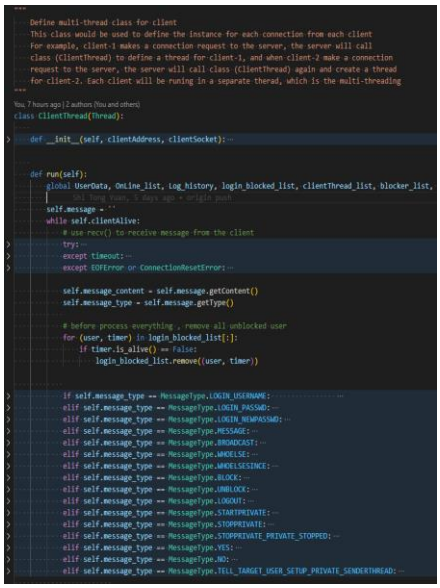
```python
class PrivateReceiveThread(Thread):
    def __init__(self, private_socket:socket): ···

    def run(self):
        self.private_socket.listen()
        self.target_socket, self.target_address = self.private_socket.accept()

        self.message = ''
        while True:
            try:
                data = self.target_socket.recv(1024)
                self.message = pickle.loads(data)
            except Exception:
                break
            print(f"[private] {username} : {self.message.getContent()}")

    def getSocket(self):
        return self.private_socket

    def endSocket(self):
        self.private_socket.close()
```

```python
class PrivateSendThread(Thread):
    def __init__(self, private_socket:socket): ···

    def run(self): ···

    def sendmessage(self,message:str):
        self.private_socket.sendall(pickle.dumps(message))

    def getSocket(self):
        return self.private_socket

    def endSocket(self):
        self.private_socket.close()
```

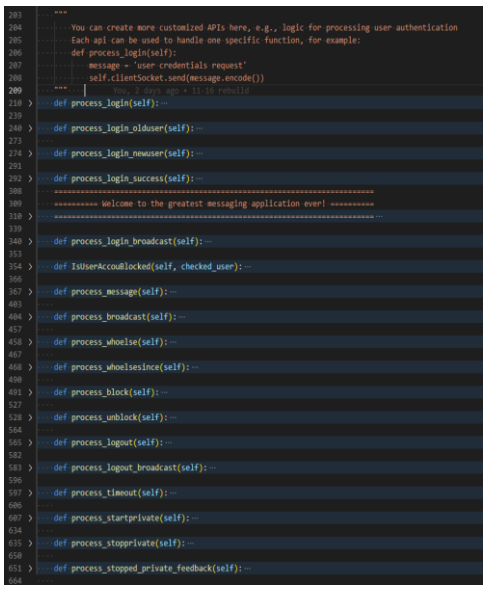
Server:

For the server, as the diagrams shown, similar to what happened in client, the server will receive the client message and then process via different MessageTypes. And there is no process function for the “private” command cause the private message go through the private socket in client, while for “startprivate” and “stopprivate” commands, the server will help the message forwarding and communication between two clients.

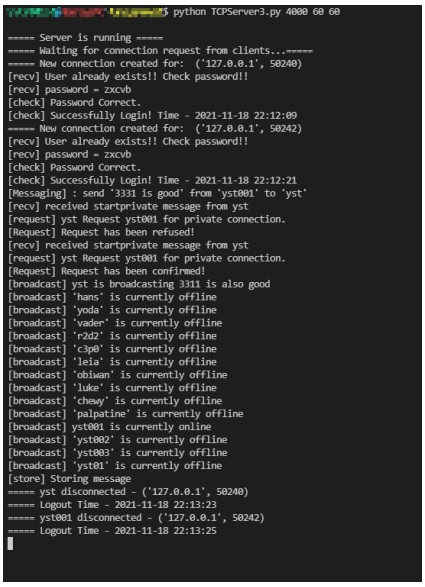
For the third image, it shows how server will be working like. As it processing the messages, it will also print information with different flag in the terminal.



```
"""
    Define multi-thread class for client
    This class would be used to define the instance for each connection from each client
    For example, client-1 makes a connection request to the server, the server will call
    class (ClientThread) to define a thread for client-1, and when client-2 make a connection
    request to the server, the server will call class (ClientThread) again and create a thread
    for client-2. Each client will be runing in a separate therad, which is the multi-threading
    """
You, 7 hours ago | 2 authors (You and others)
class ClientThread(Thread):

    def __init__(self, clientAddress, clientSocket):...


    def run(self):
        global UserData, OnLine_list, Log_history, login_blocked_list, clientThread_list, blocker_list,
                You, 5 days ago • origin push
        self.message = ''
        while self.clientAlive:
            # use recv() to receive message from the client
            try:...
            except timeout:...
            except EOFError or ConnectionResetError:...

            self.message_content = self.message.getContent()
            self.message_type = self.message.getType()

            # before process everything , remove all unblocked user
            for (user, timer) in login_blocked_list[:]:
                if timer.is_alive() == False:
                    login_blocked_list.remove((user, timer))


            if self.message_type == MessageType.LOGIN_USERNAME:...
            elif self.message_type == MessageType.LOGIN_PASSWD:...
            elif self.message_type == MessageType.LOGIN_NEWPASSWD:...
            elif self.message_type == MessageType.MESSAGE:...
            elif self.message_type == MessageType.BROADCAST:...
            elif self.message_type == MessageType.WHOELSE:...
            elif self.message_type == MessageType.WHOELSESINCE:...
            elif self.message_type == MessageType.BLOCK:...
            elif self.message_type == MessageType.UNBLOCK:...
            elif self.message_type == MessageType.LOGOUT:...
            elif self.message_type == MessageType.STARTPRIVATE:...
            elif self.message_type == MessageType.STOPPRIVATE:...
            elif self.message_type == MessageType.STOPPRIVATE_PRIVATE_STOPPED:...
            elif self.message_type == MessageType.YES:...
            elif self.message_type == MessageType.NO:...
            elif self.message_type == MessageType.TELL_TARGET_USER_SETUP_PRIVATE_SENDERTHREAD:...
```



```
    """
    You can create more customized APIs here, e.g., logic for processing user authentication
    Each api can be used to handle one specific function, for example:
    def process_login(self):
        message = 'user credentials request'
        self.clientSocket.send(message.encode())
    """     You, 2 days ago • 11-16 rebuild
    def process_login(self):...

    def process_login_olduser(self):...

    def process_login_newuser(self):...

    def process_login_success(self):...
    ================================================================
    ========== Welcome to the greatest messaging application ever! ==========
    ================================================================...

    def process_login_broadcast(self):...

    def IsUserAccouBlocked(self, checked_user):...

    def process_message(self):...

    def process_broadcast(self):...

    def process_whoelse(self):...

    def process_whoelsesince(self):...

    def process_block(self):...

    def process_unblock(self):...

    def process_logout(self):...

    def process_logout_broadcast(self):...

    def process_timeout(self):...

    def process_startprivate(self):...

    def process_stopprivate(self):...

    def process_stopped_private_feedback(self):...
```



```
[redacted]$ python TCPServer3.py 4000 60 60

===== Server is running =====
===== Waiting for connection request from clients...=====
===== New connection created for:  ('127.0.0.1', 50240)
[recv] User already exists!! Check password!!
[recv] password = zxcvb
[check] Password Correct.
[check] Successfully Login! Time - 2021-11-18 22:12:09
===== New connection created for:  ('127.0.0.1', 50242)
[recv] User already exists!! Check password!!
[recv] password = zxcvb
[check] Password Correct.
[check] Successfully Login! Time - 2021-11-18 22:12:21
[Messaging] : send '3331 is good' from 'yst001' to 'yst'
[recv] received startprivate message from yst
[request] yst Request yst001 for private connection.
[Request] Request has been refused!
[recv] received startprivate message from yst
[request] yst Request yst001 for private connection.
[Request] Request has been confirmed!
[broadcast] yst is broadcasting 3311 is also good
[broadcast] 'hans' is currently offline
[broadcast] 'yoda' is currently offline
[broadcast] 'vader' is currently offline
[broadcast] 'r2d2' is currently offline
[broadcast] 'c3p0' is currently offline
[broadcast] 'leia' is currently offline
[broadcast] 'obiwan' is currently offline
[broadcast] 'luke' is currently offline
[broadcast] 'chewy' is currently offline
[broadcast] 'palpatine' is currently offline
[broadcast] yst001 is currently online
[broadcast] 'yst002' is currently offline
[broadcast] 'yst003' is currently offline
[broadcast] 'yst01' is currently offline
[store] Storing message
===== yst disconnected - ('127.0.0.1', 50240)
===== Logout Time - 2021-11-18 22:13:23
===== yst001 disconnected - ('127.0.0.1', 50242)
===== Logout Time - 2021-11-18 22:13:25
```