CSEE4119 Super Chatter

Name: Qing Lan UNI: ql2282

Description

Super Chatter is a Chatting application built in python what enable several users chatting together. It used the UDP protocol + multi-threading to achieve the full functionalities for clients and server.

Tutorial

To provides the full functionalities of the program and avoid unexpected error please run the followings:

```
$ sudo apt-get install build-essentials
```

To run the application, please move to the folder of the Python file. If client has been set-up before server, the client would automatically quit.

Run as a server

```
$ python UdpChat.py -s <port>
```

Run as a client

```
$ python UdpChat.py -c <nick-name> <server-ip> <server-port> <client-port>
```

Startup

An Operating Server will tipically show:

```
dyn-160-39-140-44:chatter lanking$ python UdpChat.py -s 5000
Socket created and binded
listening on: localhost : 5000
Server Mode Start
```

For Client Side:

```
dyn-160-39-140-44:chatter lanking$ python UdpChat.py -c sonic localhost 5000 1211
Socket created and binded
Client Mode Start
>>> [Welcome, You are registered.]
>>> [Client table updated.]
>>>
```

Chatting

Send a Chat Message (From Lanking to sonic)

Lanking

```
>>> [Welcome, You are registered.]
>>> [Client table updated.]
>>> [Client table updated.]
>>> send sonic hi
>>> [Message received by sonic.]
>>>
```

Sonic

```
>>> [Welcome, You are registered.]
>>> [Client table updated.]
>>> lanking: hi
>>>
```

Offline Chatting

Case: sonic shutdown (No Dereg)

Message shown on Lanking's Terminal

```
>>> send sonic hi
>>> [No ACK from sonic, message sent to server.]
[Messages received by the server and saved]
>>> [Client table updated.]
>>>
```

And a File namely "sonic" will be generated with the following content

```
{"Msg": "hi", "Sender": "lanking", "Time": 1488400115.368341}
```

If sonic Log-back, the file will be loaded properly and removed.

```
>>> reg sonic
>>> [Welcome, You are registered.]
>>> [Client table updated.]
>>> [You Have Messages]
>>> lanking: 2017-03-01 15:28:35 hi
>>>
```

Login and Logoff

Logoff

```
>>> dereg
>>> [You are Offline. Bye.]
```

Login

```
>>> reg lanking
>>> [Welcome, You are registered.]
>>> [Client table updated.]
```

Program Feature

This application is designed for Online/Offline communication with complete login/logoff architecture. The protocol this App use is the UDP connection. It contains the a comprehensive error handling features for the know issues:

- Accident Offline handling: Client notify Server -> Server store offline message and update table
- Conflict Login: Username should be unique while the server operating
- Dead Online Client: Server has a Verification function used to check the online status of the client
- Infinity Acked: Avoid infinity Acknowledge message sent among clients and server
- Server Offline: When a server offline, it will notify all the clients to go offline too
- Thread Safe: Set Daemon on Listener and Sender, keep Main thread handle the Interruption

The application improves the Ctrl+C interruption handling. It would logoff automatically if Ctrl+C pressed. Please shut down the terminal to create the Dead-Online-Client Scenario. Same way works on the server if Dead-Online-Server Scenario needed.

Data Structure and Internal Logic

The whole application were built based the two classes, Server and client. All messages are parsed and loaded in JSON.

Server

Variable

- addrbook: A dictionary used to store updated Database
- kill: A Boolean flag used to shut down the listener of the Server
- s: A Socket used to listen and send message

Function

- **init** : Used to create new socket and bind to the IP and port

- listen : Listening logic for server contains functionality to send ACK in response
- send: Send the message to the client (Contains ACK fetcher to fetch ACK)
- broadcast: Send information to all available user
- Update_table: Major function to update the registration table, dealing with registration request
- dereg_user: Process Deregistration request from Client
- verification: Check the user is online or not
- offline_msg: Dealing with offline request and store messages

Accepted Request

- Reg: Registration Request
- Dereg: Dereg Request
- Offline: Offline Storage Request
- ack: For acknowledgement

Client

Variable

- addrbook: A dictionary used to store updated Database
- kill: A Boolean flag used to shut down the listener and sender of the Client
- acked: Acknowledge boolean flag created to confirm a message has been received.
- nickname: The current nickname for the client
- Serverip: IP address of the server
- Serverport: Port of the server
- PORT: Port used for the client

Function

- init: Used to create new socket and bind to the IP and port
- listen: Listening logic for server contains functionality to send ACK in response and set acked flag
- send: Send the data with 500msec timeout with acked flag check
- updatetable: Update the addresstable
- reg: Registration request sent from client
- serversend: 5 times checking-send to the server
- dereg: Deregistration request
- suicide: Handle the server-no-response condition, the client suicide

Accepted Request

- Chat: Chat Message sent from Server or client
- Table: Table Update request sent from server
- Notification: Notification Message sent from server
- Verification: Verification Request sent from server
- ack: For acknowledgement

Main Logic

Server

Server Will runs in a Loop switching between different cases

Client

Client Listener will runs in a loop switching between different cases and print out message.

Client Sender will take input from user and determine the condition of the client

Current Known Issue

Handle Multiple request at the same time

The client and server's listener are designed with 100 msec timeout. Considering a large amount of users using the application at the same time, there maybe some packet timed-out. This issue could be fixed in the future version by adding a Complex threading system. Every Listener itself is a thread with an expire time. The listening process wouldn't delayed by a single thread. If is possible to use a queue to process the incoming requests.

Ctrl + C Triggering

The handling of Ctrl + C will work all the time. However, sometimes you need to Ctrl + C + Enter to Get it proceed.

Data Present Imcomplete

When Trying to type in Chinese -- Japanese Characters, the message received are sometimes imcomplete