Rapport Réseau 2019

Groupe TM51-1

April 2019

Table des matières

Introduction

Il s'agit d'implémenter un serveur/client chat.Internet Relay Chat ou IRC (en Français, "discussion relayée par Internet") est un protocole de communication textuelle sur Internet. Il sert a la communication instantanée principalement sous la forme de discussion en groupe par l'intermédiaire de canaux de discussion, mais peut aussi être utilisé pour de la communication de un à un. Il peu par ailleurs être utilisé pour du transfert de fichiers. Il a été concu fin Aout 1988 l'IRC a été décrit initialement dans la RFC 1459.

Le protocole que nous avons mis en place, permet parfaitement de réaliser l'ensemble des tâches demandées. Pour commencer, il y a la gestion des connexions et des déconnexions des clients au serveur. Ces fonctionnalités sont gérées par notre protocole tout en tenant compte des erreurs potentielles comme la déconnexion brusque d'un client ou encore une indisponiblité du serveur. Ces erreurs sont gérées avec rigueur : par exemple si le serveur est fermé de façon brusque avec un "ctrl + c", on déconnecte tous les autres clients et lorsque le réseaux est interrompue entre le serveur et un seul client, le serveur déconnecte le seul client et permet donc aux autres de continuer les discussions.
Par ailleurs le projet contient deux versions : la version minimale V0 et la version étendue V1.

Chapitre 1

version minimale V0

1.1 Description

Il s'agit là d'implémenter une version minimale du client/serveur chat. Le serveur gère différentes conversation dans les canaux dédies (”channel” en IRC), un client peut communiquer dans un seul canal. toutes les commandes sont préfixer avec '/' (mais le caractère '/' ne dois pas être envoyé sur le réseau)
Les differentes commandes sont :

```
* /HELP: print this message
* /LIST: list all available channels on server
* /JOIN <channel>: join (or create) a channel
* /LEAVE: leave current channel
* /WHO: list users in current channel
* <message>: send a message in current channel
* /MSG <nick> <message>: send a private messag
* /BYE: disconnect from server
* /KICK <nick>: kick user from current channel
* /REN <channel>: change the current channel n
```

1.2 Implémentassions

Le projet a été implémenter en python, et il contient deux fichiers : server.py et client.py. L'implémentassions que nous avons mis en place utilise le protocole connecté TCP et le module select. Tout d'abord coté serveur nous avions fait appel a la méthode select pour récupérer la liste des sockets(client) qui veut se connecter , le serveur fait un bind() sur le port 1459 pour se greffer au port et attendre que les clients se connectes. Ensuite tout client qui voudra se connecté se connecte sur ce port et en localhost.

Tout d'abord lors d'une nouvelle connexion au serveur , il ya un échange de données entre le client et le serveur.Dès que le client se connecte le serveur l'envoie un message lui demandant de s'identifier avant de continuer.Pour cela on a utiliser un dictionnaire nommer "ladress" qui a comme clé la socket du client et valeur la chaîne de caractère que le client aurai rentré et on gère dans une fonction appelée nick-client avec pour parametre la chaine que le client a rentré.Le principe de cette fonction est la suivante le client se connecte le serveur lui demande son nick et d'bord le serveur teste réellement si la socket du client se trouve dans la liste de socket qu'il détient ensuite le client tape une chaîne le serveur accepte n'importe quelle chaîne excepté une chaîne qui commence par '/' car le serveur considère toute chaîne commençant par '/' et le commencement d'une commande or le client n'a pas le droit d'utiliser une commande avant de s'identifier ensuite le serveur ajoute ce élément dans notre dico ladress.

```
def nick_client(sock_nick, nick):
    if nick[0] == "/":
        gestionMessage(connection, "nick
    else:
        laddress[connection] = nick
```

Ensuite le serveur demande au client connecté de joindre un canal de communication , pour cela on a géré dans une fonction 'rejoindre-Canal' afin de d'implémenter on a rajouté deux dictionnaires clients-channels et member-in-channels, le premier est un dictionnaire qui a comme clé le nick du client et comme valeur son canal correspondant et le deuxième a comme clé une liste de tous les client qui vont se joindre a ce canal. Il s'agit là de vérifier si le client ne se trouve dans un canal et si le canal qu'il a tapé existe on lui rajoute dans le canal en le rajoutant dans la liste des membre du canal(member-in-channel) tout en le rajoutant dans le dico client-channel , si le canal n'excite pas le client

le crée juste en rajoutant et s'ajoutant dedans comme administrateur du canal sans oublier de le rajouter dans le dico client-channel.

Une fois dans un canal le client est libre d'utiliser parmi les commandes cité en haut exception faites hormis l'administrateur d'un canal aucun client n'a le droit d'utiliser les commandes : 'REN' pour renommer un canal et 'KICK' pour éjecter un client. On a implémenter différentes fonction afin de gérer ces commandes et toutes ces commandes sont gérées dans un fonction appelée 'gestionCommande' et exécute dans la une boucle.

```
while True:
    connection_demandee, wlist, rlist = select.select(socketl+[scket], [], [])
    for connection in connection_demandee:
        if (connection == scket):
            connection , infoclient = connection.accept()
            gestionMessage(connection, "please enter another nick \n")
            socketl.append(connection)
        else:
            msgclient = connection.recv(1500)
            data = msgclient.decode()
            #message vide
            if (len (data) == 0):
                sedeconnecter(connection)

            else:
                if connection not in laddress:
                    client = data[:-1]
                    if client in clients_channels:
                        gestionMessage(connection, "please enter another nick \n"
                    else:
                        nick_client(connection, client)

                else:
                    gestionCommande(connection, data)
```

Toutes les commandes sont délimitées par un '/' et il des fonctions qu'on a implémenter pour comme : getadress qui prend en paramètre un client et renvoi sa socket correspondant cette fonction nous a permis de réussir sur plusieurs fonction notamment pour l'envoi des messages, ensuite la fonction getAdmin avec comme paramètre un canal et renvoie l'administrateur de ce canal qui servit dans les commande 'KICK' et 'REN'.

Pour la gestion des messages , tout ce qu'un client tape est un string ainsi nous avons un seul recv coté serveur qui reçois le message et traite selon les demandes du client avant de le renvoyer au client et on une fonction gestionMessage qui prend en paramètre un socket et un string l'encode et l'envoie au socket a l'aide de sendall().

Chapitre 2

Version étendue V1

2.1 Description

L'idée est la même il s'agit bien d'implémenter un client/server chat l'étendue de la version minimale. Ici un client peut se connecter dans plusieurs canaux
Avec la modification de la version minimale s'ajoute aussi certaines dictionnaire comme : current-channel qui a comme clé le nick du client et comme valeur le canal courant du client, le dico channel-Admin qui a comme clé un channel et valeur l'ensemble de ces administrateurs et un dictionnaire client-contenu qui a comme clé le nick du client qui va doit recevoir le fichier et comme valeur le fichier pour le send et recv des fichiers. Par ailleurs clients-channels sur la V0 devient un dico qui a comme clé un client et comme valeur la liste de ses canaux.

```
* /CURRENT: print current channel name
* /CURRENT <channel>: set current channel
* /MSG <nick1;nick2;...> <message>: send a p
* /NICK <nick>: change user nickname on serv
* /GRANT <nick>: grant admin privileges to a
* /REVOKE <nick>: revoke admin privileges [a
* /SEND <nick> </path/to/file>: send a file
* /RECV </path/to/file>: receive a file and
* /HISTORY: print history of current channel
```

Chapitre 3

Conclusion

Avec la version finale seule la commande HISTORY qu'on a pa eu le temps d'implémenter,l'ensemble des autres fonction marche et a été tester sur moodle.

3.1 Conclusion de Thierno Amadou Diallo

Dans l'ensemble ce projet a été d'une véritable expérience car chaque partie du projet m'a poussé a faire des recherches et des analyses profondes. Il m'a aussi permit de comprendre le fonctionnement du protocole TCP.

3.2 Conclusion de Rassidata Barry

En conclusion, je dois avouer que rétrospectivement nous somme satisfaits de ce projet puisque nous avons atteint des nombreux objectifs.

En effet,ce mini projet m'a permis de comprendre et apprendre la communication qui est entre un serveur et un client avec l'utilisation de beaucoup de commandes à l'aide du langage python, ce projet m'a aussi permis de connaître les bases préliminaires de la programmation en langage python.

Pour finir nous remercierons nos encadreurs qui se sont véritablement occupés de nous et qui étaient toujours prêt à répondre à chacune de nos questions.

Grille d'auto-évaluation

Version 0

Commandes	Score (/10)
connect & disconnect without errors	10
login & bye (/BYE)	10
list channels (/LIST)	10
join & leave channel (/JOIN & /LEAVE)	10
user list (/WHO)	10
public message	10
private message (/MSG)	10
rename channel (/REN)	10
kick user (/KICK)	10

Version 1

Commandes	Score (/10)
change current channel (/CURRENT)	10
private message to several users (/MSG)	10
change user nickname (/NICK)	10
grant admin privileges (/GRANT)	10
revoke admin privileges (/REVOKE)	10
send & receive file (/SEND & /RECV)	10
history (/HISTORY)	0

Bonus (à compléter)

Commandes	Score (/10)
bonus 1 (/CMD1)	
bonus 2 (/CMD2)	
bonus 3 (/CMD3)	
...	