Compte rendu R309 Aurélien BESOMBES RT2-FI

-- Server --

I: Implémentation des modules & variable de base ~

```
import socket
import threading
import json
#---------------------------------------------------------------------------
localIP     = ""
localPort   = 5000
bufferSize  = 1024
TCPServerSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
TCPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
TCPServerSocket.bind((localIP, localPort))

print("TCP server up and listening")
TCPServerSocket.listen(100)

non_secure = {'admin':'admin','user1':'user1'}
```

- **non_secure** → dictionnaire avec lien nom d'utilisateur et mot de passe.

II: Class pour chaque utilisateur et channel ~

```
class User():
    def __init__(self,conn,addr):
        self.conn = conn
        self.addr = addr
        self.us = None
        self.channel = None

    def get_conn(self): ...
    def get_addr(self): ...
    def get_us(self): ...
    def get_channel(self): ...
    def change_us(self,new_us): ...

    def change_channel(self,new_channel):
        if new_channel in exist_channel:
            self.channel = exist_channel[new_channel]
            exist_channel[new_channel].add_user(self)
        elif self.channel == None:
            add_channel = Channel(str(new_channel))
            exist_channel[str(new_channel)] = add_channel
            self.change_channel(new_channel)
        elif self.channel != None:
            self.channel.remove_user(self)
            add_channel = Channel(str(new_channel))
            exist_channel[str(new_channel)] = add_channel
            self.change_channel(new_channel)
        else:
            pass

class Channel():
    def __init__(self,name):
        self.name = name
        self.list_user = []

    def get_name(self): ...
    def get_list_user(self): ...

    def add_user(self,user_add):
        if user_add not in self.list_user:
            self.list_user.append(user_add)

    def remove_user(self,user_del):
        self.list_user.remove(user_del)

home = Channel('home')
exist_channel = {'home':home}
```

- Pour chaque utilisateur connecter, on créer un object avec comme information, la connexion et l'addr donné par le socket, mais aussi le nom d'utilisateur et le channel (object).
- Pour changer le channel de l'utilisateur, on vérifie si le channel existe déjà dans la variable `exist_channel`, si oui, on le rajoute dedans et le retire de son ancien channel. Sinon, on créer un nouveau channel avec la class `Channel` et on le rajoute dans `exist_channel`.

III: Boucle pour chaque nouvelle connection ~

```
while True:
    conn, addr = TCPServerSocket.accept()
    user_add = User(conn,addr)

    exist_channel['home'].add_user(user_add)
    user_add.change_channel('home')

    print (addr[0] + " connected")
    start = threading.Thread(target=clientthread,args=[user_add])
    start.start()
```

- Lors d'une connexion, un object est créer avec les informations données par Socket et on rajoute cet utilisateur dans le channel "Home" qui est celui par défaut. Puis un `Thread` est créer pour gérer l'utilisateur.

IV: Fonction pour gérer les messages et les utilisateurs ~

```
def send_message(channel_thread,data):
    u_del = []
    for c in channel_thread.get_list_user():
        try:
            c.get_conn().sendall(data.encode())
        except:
            u_del.append(c)

    if len(u_del) != 0:
        for user_del in u_del:
            channel_thread.remove_user(user_del)

def clientthread(user_thread):
    conn = user_thread.get_conn()
    addr = user_thread.get_addr()
    channel_thread = user_thread.get_channel()
    while True:
        try:
            data = conn.recv(1024).decode()
            data = json.loads(data)
            if 'SEND' in data:
                mess = data['SEND']['PAYLOAD']
                user = data['SEND']['USER']
                channel_thread = user_thread.get_channel()
                data = f'{user_thread.get_us()}>{mess}'
                send_message(channel_thread,data)

            elif 'CONN' in data:
                us = data['CONN']['us']
                pwd = data['CONN']['pwd']

                if us.startswith('Gue'):
                    conn.send("OK".encode())
                    user_thread.change_channel(data['CONN']['channel'])
                    user_thread.change_us(us)
                    send_message(user_thread.get_channel(),f"[+]{user_thread.get_us()}")

                else:
                    if us in non_secure and non_secure[us] == pwd:
                        user_thread.change_us(us)
                        send_message(user_thread.get_channel(),f"[+]{user_thread.get_us()}")
                    else:
                        conn.send("NO".encode())

            elif 'END' in data:
                send_message(channel_thread,f"[-]{user_thread.get_us()}")
                conn.close()

            else:
                conn.close()
                channel_thread.remove_user(user_thread)
```

```
        except OSError:
            pass
```

- Les messages envoyés sont au format JSON contenant les informations permettant de savoir s'il s'agit d'un message ('SEND'), d'une authentification ('CONN') ou de la fermeture de session ('END').
- Lors de l'authentification ('CONN') l'utilisateur envoie son login/mdp et le channel (si ce n'est pas "home", il en créer un nouveau et le nom donné).
- Lors d'un envoi de message, on appelle la fonction "send_message" qui envoie un message à tous les utilisateurs présent sur le même channel que l'envoyeur. De plus, cette fonction permet de vérifier si l'envoie à marché et si ce n'est pas le cas, il supprime l'utilisateur qui est considéré comme 'mal déconnecté'.

-- Client --

I: envoie du login/mdp et channel ~

```
info_for_chat = {'def_channel': 'home', 'def_user' : ''}

def headler(sig, frame):
    disconnect()

def on_closing():
    disconnect()

def disconnect():
    stop = json.dumps({'END': {}})

    TCPClientSocket.send(str.encode(stop))
    TCPClientSocket.close()
    fenetre.destroy()
    exit_event.set()


def main():
    global info_for_chat
    def send_mess():
        global info_for_chat
        if Tchnl.get() == '':
            info_for_chat['def_channel'] = 'home'
        else:
            info_for_chat['def_channel'] = Tchnl.get()

        if Tusr.get() != str(''):
            info_for_chat['def_user'] = Tusr.get()
        else:
            info_for_chat['def_user'] = f'Guest{random.randint(100, 999)}'

        mess = json.dumps({'CONN':
{'us':info_for_chat['def_user'],'pwd':Tpwd.get(),'channel':info_for_chat['def_channel']}})
        TCPClientSocket.send(str.encode(mess))
        login()

    def login():
        data = TCPClientSocket.recv(1024).decode()
        if data != 'NO':

            #creation de tout les widgets avec tkinter
            # ...

            when_login()
        elif data == 'NO':
            print("Mdp non valide")
        else:
            print("User déjà connecté")
```

- La fonction disconnect permet de fermer correctement la connexion au serveur et le programme client.
- La fonction main possède deux fonctions. La première permet de récupérer les informations de connexion dans les différents widgets de Tkinter. S'ils sont vide, on créer un profil 'Guest'.
- Si les champs ne sont pas vide, on envoie les identifiants au serveur, et si le serveur nous envoie "NO", c'est que les identifiants ne sont pas bons. Sinon, on accède au serveur.

II: Attente des messages venant du server ~

```
def when_login():
    def envoie(event):
        mess = json.dumps({'SEND':
{'USER':info_for_chat['def_user'],'CHANNEL':info_for_chat['def_channel'],'PAYLOAD':Tinput.get()}})

        TCPClientSocket.send(str.encode(mess))
        Tinput.delete(0,"end")
        Tchat.see("end")

    def check():
        while True:
            try:
                data = TCPClientSocket.recv(1024).decode()
                Tchat.insert(END, f'{data} \n' )
            except ConnectionAbortedError:
                break

            except ConnectionResetError:
                break

    #Creéation des widgets avec tkinter
    # ...

    start = threading.Thread(target=check)
    start.start()
```

- On définit deux fonctions permettant d'envoyer un message et l'autre d'attendre qu'un message soit reçu depuis le serveur.
- Cette deuxième fonction est lancée avec un thread.
- La fonction envoie est un callback de tkinter permettant d'envoyer un message au serveur. (Le message sera ensuite affiché par le fait que le serveur enverra le message à tout le monde, dont nous)

III: Lancement général du programme client ~

- Dès qu'un message est reçu du serveur, il affiche dans le widget fait pour, dans l'interface Tkinter. Cette partie est gérée par un thread.

```
main()

fenetre.protocol("WM_DELETE_WINDOW", on_closing)
#fermeture avec la croix

signal.signal(signal.SIGINT, headler)
#fermeture avec ctrl+c

fenetre.mainloop()
```

- On lance la fonction main et on définie des actions à faire si on faire la fenêtre tkinter et si on fait ctrl+c dans le terminal.