Cryptographie

Projet de tchat TCP

Introduction :

Le but de ce projet est de developper un tchat TCP en python. Il s'agit donc de créer un système de messagerie par lequel vous pourrez communiquer.
Nous allons donc voir étape par étape comment coder un serveur par lequel transiteront vos messages. Puis nous coderons le client qui nous servira à se connecter au serveur pour envoyer nos messages et en recevoir.

Partie 1 : Les prérequis

Pour faire ce projet, deux bibliothèques vont nous être utiles :

```
1  import socket
2  import threading
```

Informations :
la bibliothèque **socket** permet de gérer facilement des clients et serveurs.
la bibliothèque **threading** permet de lancer plusieurs fonctions en même temps comme si ces dernières étaient indépendantes.

Partie 2 : Le serveur

Etape 1 — La base de notre serveur

1. Créer un fichier **serveur.py**
2. Importez y nos deux librairies **socket** et **threading**

Pour que notre serveur fonctionne, il doit être à l'écoute sur certains ports. Pour se faire, nous allons utiliser un socket, un socket va nous permettre d'envoyer et recevoir des informations sur le réseau.

3. Parametrage de notre socket

```
1  host = '0.0.0.0'
2  port = 5555
3
4  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5  server.bind((host, port))
6  server.listen()
```

L'adresse ip **0.0.0.0** permet au serveur d'écouter toutes les adresses ip pour nous faciliter la tâche. Le port **5555** est le port par lequel nous comuniquerons.
Les trois lignes de codes suivantes servent à créer notre serveur et à le "lancer".

4. Sauvegarde des utilisateurs

Pour des fonctionalités futures, nous allons avoir besoin de conserver nos utilisateurs connectés. Créer donc unu liste **clients** vide ainsi qu'une liste **pseudos** vide.

Etape 2 _____ Les fonctions de notre serveur

Notre fichier **serveur.py** devra posséder 3 fonctions :

— diffuser() qui sert à envoyer les messages

— gestion() qui sert à faire transiter les messages et gérer les deconnexions

— connexion() qui sert à écouter et connecter les nouveaux clients

Fonction diffuser() :

Completer la fonction diffuser() servant à envoyer les messages à tout le monde sauf au client qui à lui même envoyé le message.

```
    def diffuser(message, client_envoyeur=None):
        for ... in ...: # Parcours de tous les client
            if ... # Verification, on envoie que si ce n'est pas le client envoyeur
                client.send(...)
```

Fonction gestion() :

Nous allons ensuite nous occuper de la fonction de "gestion du serveur".
Cette dernière va écouter en permanance si un message est reçu, puis le traiter. Dans cette fonction, si l'on capte qu'un client n'est plus joignable, on le deconnecte, puis on le retire des listes **clients** et **pseudos**
Complétez la pour qu'elle soit opérationelle.

```
    def gestion(client):
        ... : # Boucle infinie pour ecouter en permanence
            try:
                message = client.recv(1024)
                diffuser(message, client)
                print(message)
            except:
                index = clients.index(client) # recuperation de l'index du client
                ... # retirer le client de la liste clients
                client.close() # deconnexion du client
                user = pseudos[index]
                diffuser(f'SERVEUR : {user} a quitte le chat'.encode('utf-8'))
                ... # retirer le pseudo du client de la liste pseudos
                break
```

Fonction connexions() :

Notre fonction connexion sert à écouter si un client tente de se connecter. Il notifie sa connexion aux autres utilisateurs puis enregistre ses informations.

```
    def connexions():
        ... : # Boucle infinie pour ecouter en permanence
            client, address = server.accept() # recuperation des informations du client
            print(f'SERVEUR : Connexion etablie avec {str(address)}')
            pseudo = client.recv(1024).decode('utf-8') # La premiere chose qu'enverra le client est son pseudo, nous pouvons donc traite ce premier messsage
            ... # ajouter le client et son pseudo aux liste clients et pseudos
            ...
```

```
8
9               print(f'SERVEUR : Nouvel utilisateur {pseudo} connecte')
10              diffuser(f'SERVEUR : {pseudo} a rejoint le
                    chat'.encode('utf-8'))
```

Tout écouter en même temps :

Maintenant que nous avons deux fonctions qui écoutent si des messages sont reçus en permanence et une écoutant si des clients tentent de se connecter.
Nous devons faire en sorte qu'elle fonctionne toutes les deux en même temps sans se marcher dessus.
Pour se faire, nous allons utiliser un **thread** pour lancer nos deux fonctions séparément.
Ajouter donc ces deux lignes de codes dans la fonction connexion.

```
1     thread = threading.Thread(target=gestion, args=(client,))
2     # Creation d'un thread ayant pour cible notre fonction gestion
3     thread.start()
4     # Lancement du thread
```

Important : lancez maintenant la fonction connexions à la fin de votre fichier pour que nos fonctions se lancent :

```
1     connexions()
```

Partie 3 : Le client

Etape 1 _____ La base de notre client

1. Créer un fichier **client.py**
2. Importez y nos deux librairies **socket** et **threading**
3. Demandez à l'utilisateur son pseudo que vous mettrez dans une variable **pseudo**
4. Nous allons maintenant établir une connexion avec le serveur

```
1     client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2     client.connect(('192.168.1.54', 5555)) # modifier l'adresse ip
```

Pensez à modifier l'adresse ip avec celle de la machine faisant tourner le serveur
Aide : pour trouver l'adresse IP d'une machine, utilisez la commande **ipconfig** dans un terminal.

5. Rapellez vous, nous devons maintenant envoyer notre pseudo comme premier message

```
1     client.send(pseudo.encode('utf-8'))
```

Etape 2 _____ Les fonctions de notre client

Notre fichier **client.py** devra posséder 3 fonctions :

— envoyer() qui sert à envoyer les messages au serveur
— ecrire() qui sert à ce que l'utilisateur puisse écrire un message en permanence
— recevoir() qui sert à écouter les messages envoyés en permanence

Fonction envoyer() :

Cette fonction simple sert à envoyer un message au serveur :

```
    def envoyer(message):
        client.send(message.encode('utf-8'))
```

Fonction ecrire() :

La fonction écrire sert à demander en permanance à l'utilisateur son message pour qu'il puisse parler à n'importe quel moment.

```
    def ecrire():
        ... # faire tourner la demande a l'infinie
            message = ... # demander a l'utilisateur son message
            ... # envoyer le message
```

Fonction recevoir() :

La fonction recevoir sert à rester à l'écoute de potentiels messages reçus, puis de les afficher. On y dissocie les messages du serveur (ces derniers commencaient toujours par "SERVEUR" dans nos fonctions serveurs) des messages des autres clients.

```
    def recevoir():
        while True:
            message = client.recv(1024).decode('utf-8')
            if 'SERVEUR' in message:
                print("\n")
                print("Message du serveur :\n")
                print(message)
                print("\n")
            else:
                print(message)
```

Etape 3 — Le threading de notre client

Maintenant que nous avons nos fonctions, nous allons devoir les faire se lancer en même temps grâce aux threads.
Ajouter ces deux lignes à la fin de votre fichier et complétez les pour que les fonctions recevoir() et ecrire() se lancent conjointement :

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

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

Partie 4 : Test du tchat

Pour tester votre tchat, plusieurs machines doivent être connectées au même réseau. Une des machine lance le serveur, et les autres le client (pensez à modifier l'adresse IP dans **client.py**)
A noter : une machine peut lancer à la fois le serveur et le client.

Partie 5 : Chiffrement de nos messages

Maintenant que nous avons vu les faiblesses de notre tchat, faites en sorte que les messages transitent en étant chiffrés. N'oubliez pas le déchiffrage !

Aide : le fichier **chiffre.py** dans lequel nous codons toutes les fonctions de chiffrement depuis le début de la séquence peut vous être utile.
Mettez le dans le dossier du projet puis importez le la où vous en avez besoin.

```
1    from chiffre import *
```

Partie 6 : Pour aller plus loin :

1 Diverses fonctionalités :

— Une meilleure gestion et affichage des pseudos

2 Tkinter pour une interface graphique

Tkinter offre de nombreuses fonctionnalités pour la création d'interfaces graphiques, notamment :

— Création de fenêtres et de widgets (boutons, champs de texte, cases à cocher, etc.).

— Gestion des événements utilisateur (clics de souris, pressions de touches, etc.).

— Gestion de mises en page (grilles, empilements, etc.).

— Personnalisation des styles et des apparences des widgets.

Pour en savoir plus sur Tkinter et apprendre à l'utiliser efficacement, voici quelques ressources utiles :

— Documentation officielle de Tkinter : `https://docs.python.org/3/library/tkinter.html`

— Tutoriels et exemples : `https://python.doctor/page-tkinter-interface-graphique-python-tutori`

— Autres tutoriels : `https://info.blaisepascal.fr/tkinter/`

En utilisant ces ressources, vous serez en mesure de maîtriser Tkinter et de créer une interface graphique pour votre **client.py**.