Petrossi
Alberto
28714148

Rapport de Projet – Mini-Projet 1

(démo à la fin)

Au cours de ce projet il a fallu réaliser dans un premier temps, dans le premier exercice, un mécanisme de proxy entre un client et un serveur, connectés via TCP. Un proxy est un programme qui agit comme un serveur vis-à-vis du client, et comme un client vis à vis du serveur. Il retransmet au serveur toutes les données que le client aurait normalement transmises au serveur, et il retransmet au client toutes les réponses que le serveur aurait normalement transmises au client.

J'ai choisi de travailler avec **Visual Studio Code** pour programmer, j'ai testé progressivement mon code en l'exécutant par le biais de terminaux. J'ai effectué mes tests localement, également en utilisant une autre machine, et sur une machine virtuelle.

Pour faire fonctionner correctement le proxy, j'ai créé deux sockets dans son programme : une socket qui écoute les connexions provenant d'un client, et une socket qui se connecte au serveur. Ainsi à chaque fois que le proxy reçoit une requête de la part d'un client, il renvoie directement cette requête du côté du serveur. Ensuite si le serveur répond au proxy, le proxy transmet cette réponse au client.

```
while True:
    connectionSocket, address = clientSocket.accept()
    print("Received")

    serverSocket = socket(AF_INET, SOCK_STREAM)
    serverSocket.connect((serverName, serverPort))
    dataClient = connectionSocket.recv(2048)
    serverSocket.send(dataClient)

    dataServer = serverSocket.recv(2048)
    connectionSocket.send(dataServer)
```

Sur cette fraction de code, « serverSocket » constitue la socket qui se connecte au serveur, tandis que « connectionSocket » constitue la socket du client qui est connectée au proxy.

Pour tester si le proxy fonctionnait, j'ai essayé d'afficher sur le serveur envoyé par le client un message quelconque dans un premier temps, et l'opération a fonctionné. Par la suite j'ai également essayé de retransmettre un autre message de la part du serveur vers le client, une fois le message du client préalablement reçu, et l'opération

Petrossi
Alberto
28714148

a également fonctionné. L'utilisation de prints sur le proxy, le serveur et le client m'a permis de comprendre ce qui allait et ce qui n'allait pas au cours de mes tests (j'utilisais les prints pour afficher les données qui arrivaient sur le proxy, le serveur ou le client).

En ce qui concerne la question 4 du premier exercice, je suis resté bloqué un peu de temps : l'objectif était de connecter plusieurs clients au proxy simultanément pour que chaque client puisse faire sa requête et recevoir sa réponse en même temps. Pour cela il a fallu que je me réfère à ce qui avait été expliqué dans le cours.

Multi-Thread TCP Server

```
from threading import *
…
def handle_client(clientSocket):
    while True:
        received = clientSocket.recv(4096)
        if not received:
            clientSocket.close()
        else:
            to_send = received.decode('utf-8').upper().encode('utf-8')
            clientSocket.sendall(to_send)
while True:
        connectionSocket, address = serverSocket.accept()
        Thread(target=handle_client,args=(connectionSocket,)).start()
```

Le Multi-Thread est la technique qui m'est semblée la plus appropriée pour aborder la question. Ainsi je me suis inspiré du code ci-dessus pour tenter de connecter plusieurs clients en même temps au proxy.
Cependant je n'arrivais pas à connecter ces différents proxy au serveur, au début je n'arrivais qu'à connecter les clients au proxy simultanément, et lorsque j'effectuais une requête vers le serveur, il fallait que je la termine du côté de tous les clients connectés avant de recevoir une réponse.

Petrossi
Alberto
28714148

```python
def handle_client(clientSocket):
    while True:
        receivedClient = clientSocket.recv(4096)
        if not receivedClient:
            clientSocket.close()
        if receivedClient:
            serverSocket = socket(AF_INET, SOCK_STREAM)
            serverSocket.connect((serverName, serverPort))
            serverSocket.send(receivedClient)
            dataServer = serverSocket.recv(4096)
            clientSocket.send(dataServer)
```

La solution a été d'effectuer la connexion au serveur à chaque fois que le proxy recevait des données du côté du client, et d'ensuite lui envoyer ces données. Puis le proxy écoute la réponse du serveur et la retransmet vers le client. En utilisant la fonction ci-dessus, j'ai réussi à faire fonctionner le principe de Multi-Thread, pour avoir plusieurs clients qui effectuent des requêtes en même temps et reçoivent des réponses en même temps.

Dans l'exercice 2, il a fallu créer différents proxy ayant chacun un rôle particulier : le premier sert de cache, le second de log, et le troisième de censeur.

J'ai effectué mes tests localement et sur une machine virtuelle, mais j'ai utilisé à chaque fois un serveur que j'ai moi-même programmé en python plutôt que d'utiliser par exemple un serveur apache.

Pour la première question la solution a été d'enregistrer du côté du proxy les informations du fichier demandé par le client (donc le navigateur web) une fois que celui-ci est récupéré du côté du serveur. En effet **on enregistre dans un fichier ayant le même nom du fichier demandé le contenu de la réponse du serveur à la requête du client**. Ainsi à chaque fois que l'utilisateur redemande l'accès à ce fichier, le proxy lui fournit la réponse enregistrée directement, **sans devoir aller la récupérer une nouvelle fois du côté du serveur**.

Petrossi
Alberto
28714148

```python
filename = Path(dataClient.split(' ')[1].lstrip("/")) #fonctionnalité de cache du serveur
if(filename.is_file()):
    file = open(filename, 'rb')
    dataFile = file.read(4096)
    connectionSocket.sendall(dataFile)
    connectionSocket.close()
else:
    serverSocket = socket(AF_INET, SOCK_STREAM)
    serverSocket.connect((serverName, serverPort))
    serverSocket.send(dataClient.encode('utf-8'))
    dataServer = serverSocket.recv(4096)
    print(dataServer)
    connectionSocket.sendall(dataServer)
    file = open(filename, 'wb')
    file.write(dataServer)
    file.close()
```

Pour la deuxième question, j'ai simplement fait en sorte que le proxy enregistre dans un fichier texte les requêtes du client et réponses du serveur. Les requêtes/réponses se rajoutent aux informations déjà présentes dans le log à chaque fois. De plus grâce à « datetime », j'ai pu afficher à chaque fois l'horaire de la requête effectuée. Et pour chaque réponse, la taille du fichier est précisée. Ainsi voici comment se présente mon fichier de log :

```
Request at 22 : 25 : 36
GET /test1.html HTTP/1.1
Size of the file : 376
Response is :
HTTP/1.1 200 OK
Server : Python HTTP Server
Connection : close



<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Sorry</title>

</head>

<body>

    <h1>You don't have the rights to look at this file, sorry :/</h1>

</body>

</html>

Request at 22 : 25 : 37
GET /favicon.ico HTTP/1.1
Size of the file : 106
```

Petrossi
Alberto
28714148

Enfin pour le proxy censeur, j'ai simplement créé une liste contenant l'ensemble des fichiers à ne pas transmettre à l'utilisateur. Et si l'utilisateur demande ce fichier, le proxy transmet à l'utilisateur un fichier par défaut qui dit que l'accès au fichier en question est interdit. Sinon il va récupérer le fichier du côté du serveur.

J'ai testé les différents proxy en les enchainant (j'ai attribué un numéro de port différent à chaque proxy et au serveur), et chaque rôle a bel et bien fonctionné. Le test a toujours été effectué avec un serveur que j'ai moi-même créé.

Pour plus de détails voici la vidéo de démonstration que j'ai réalisé :

https://youtu.be/bxWx_q7C5so