

Rapport

Mini projet 1

PROGRES

21/10/2021

Binôme :

- AKLI Mohammed Farouk.
- BIRI Camelia.

Table de matière

1	Introduction :.....	3
1.1	Proxy :.....	3
1.1.1	Le principe de fonctionnement d'un proxy :.....	3
1.2	Le principe de cache :.....	3
1.2.1	Le principe de logueur HTTP :	4
1.2.2	Le principe de censeur HTTP :.....	4
2	Exercice 1 Proxy TCP :.....	4
2.1	Explication des décisions techniques :	4
2.1.1	Client :.....	4
2.1.2	Proxy :	5
2.1.3	Serveur :.....	7
2.1.4	Plusieurs clients :.....	8
3	Exercice 2 Proxy HTTP :.....	8
3.1	Explication des décisions techniques :	8
3.1.1	Client :.....	8
3.1.2	Proxy :	8
3.1.3	Serveur :.....	8
3.2	Voici un schéma d'illustration du principe de fonctionnement :	9
4	Conclusion	9

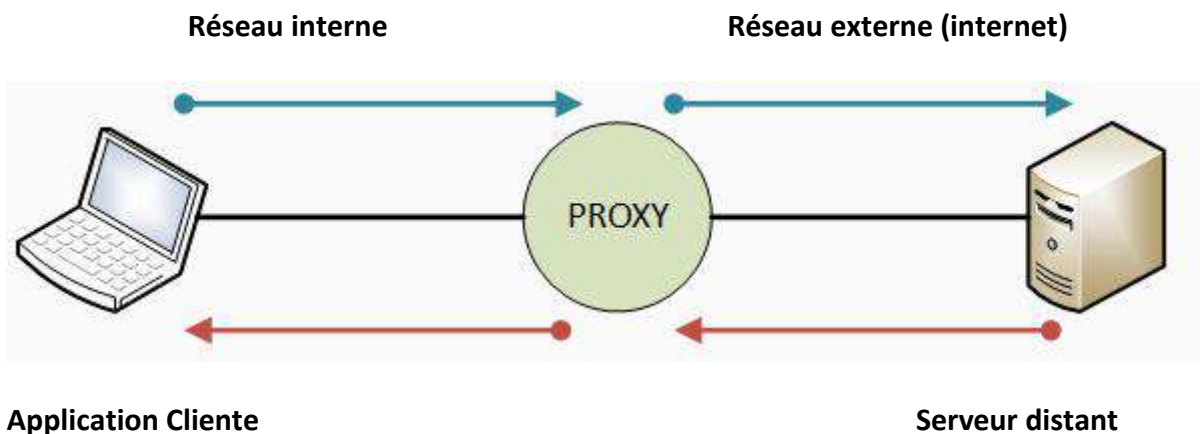
1 Introduction :

1.1 Proxy :

Un serveur proxy (en anglais «*proxy server*», appelé aussi «*serveur mandataire*») est un composant logiciel informatique faisant fonction d'intermédiaire entre les ordinateurs d'un réseau local et internet.

1.1.1 Le principe de fonctionnement d'un proxy :

Le principe de fonctionnement basique d'un serveur proxy est assez simple : il s'agit d'un serveur "mandaté" par une application pour effectuer une requête sur Internet à sa place. Ainsi, lorsqu'un utilisateur se connecte à internet à l'aide d'une application cliente configurée pour utiliser un serveur proxy, celle-ci va se connecter en premier lieu au serveur proxy et lui donner sa requête. Le serveur proxy va alors se connecter au serveur que l'application cliente cherche à joindre et lui transmettre la requête. Le serveur va ensuite donner sa réponse au proxy, qui va à son tour la transmettre à l'application cliente.



Le proxy se situe au niveau de la couche application c à d niveau 7 (HTTP, FTP, SSH, etc.). Le plus courant est le proxy Web, qui relaie donc des requêtes et des réponses HTTP.

1.2 Le principe de cache :

La plupart des proxys assurent ainsi une fonction de cache (en anglais *caching*) qui désigne le stockage temporaire des données (pages Web) fréquemment consultées, ce qui rend l'accès à l'avenir plus facile et plus rapide.

Cette fonctionnalité permet d'une part de réduire l'utilisation de la bande passante vers internet ainsi que de réduire le temps d'accès aux documents pour les utilisateurs.

Toutefois, pour mener à bien cette mission, il est nécessaire que le proxy compare régulièrement les données qu'il stocke en mémoire cache avec les données distantes afin de s'assurer que les données en cache sont toujours valides et mise à jour.

1.2.1 Le principe de logueur HTTP :

Le principe de logueur HTTP consiste à archiver dans un fichier de log, toutes les requêtes GET des clients et les réponses du serveur. Le fichier de log doit contenir suffisamment d'information pour effectuer des audits c à d pouvoir retrouver les adresses IP des clients qui ont obtenu une réponse non vide d'un serveur concernant une URI.

1.2.2 Le principe de censeur HTTP :

Le proxy dispose d'une liste de sites interdits (fournie en entrée au proxy). Si l'URI demandée dans une requête GET du client contient un lien qui renvoie vers un site interdit, ce lien est remplacé par un message « Interdit » dans le corps de la réponse.

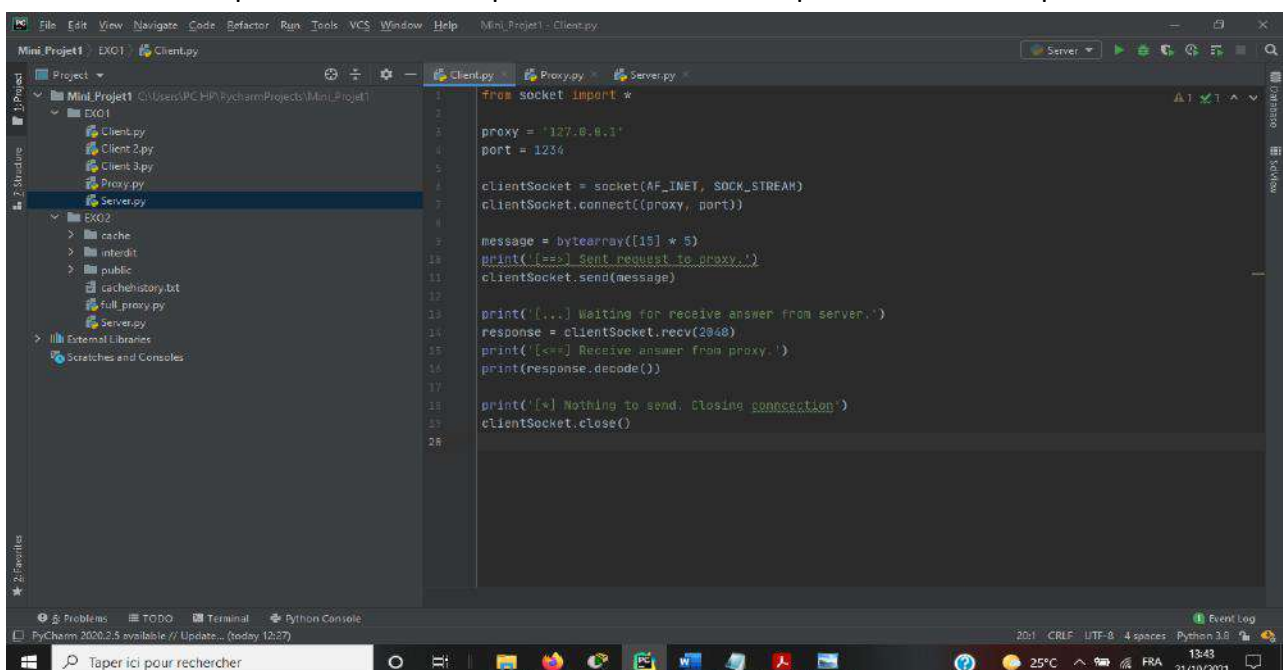
2 Exercice 1 Proxy TCP :

On cherche à programmer en Python un mécanisme de proxy entre un client et un serveur normalement connecté 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 données que le serveur aurait normalement transmises au client.

2.1 Explication des décisions techniques :

2.1.1 Client :

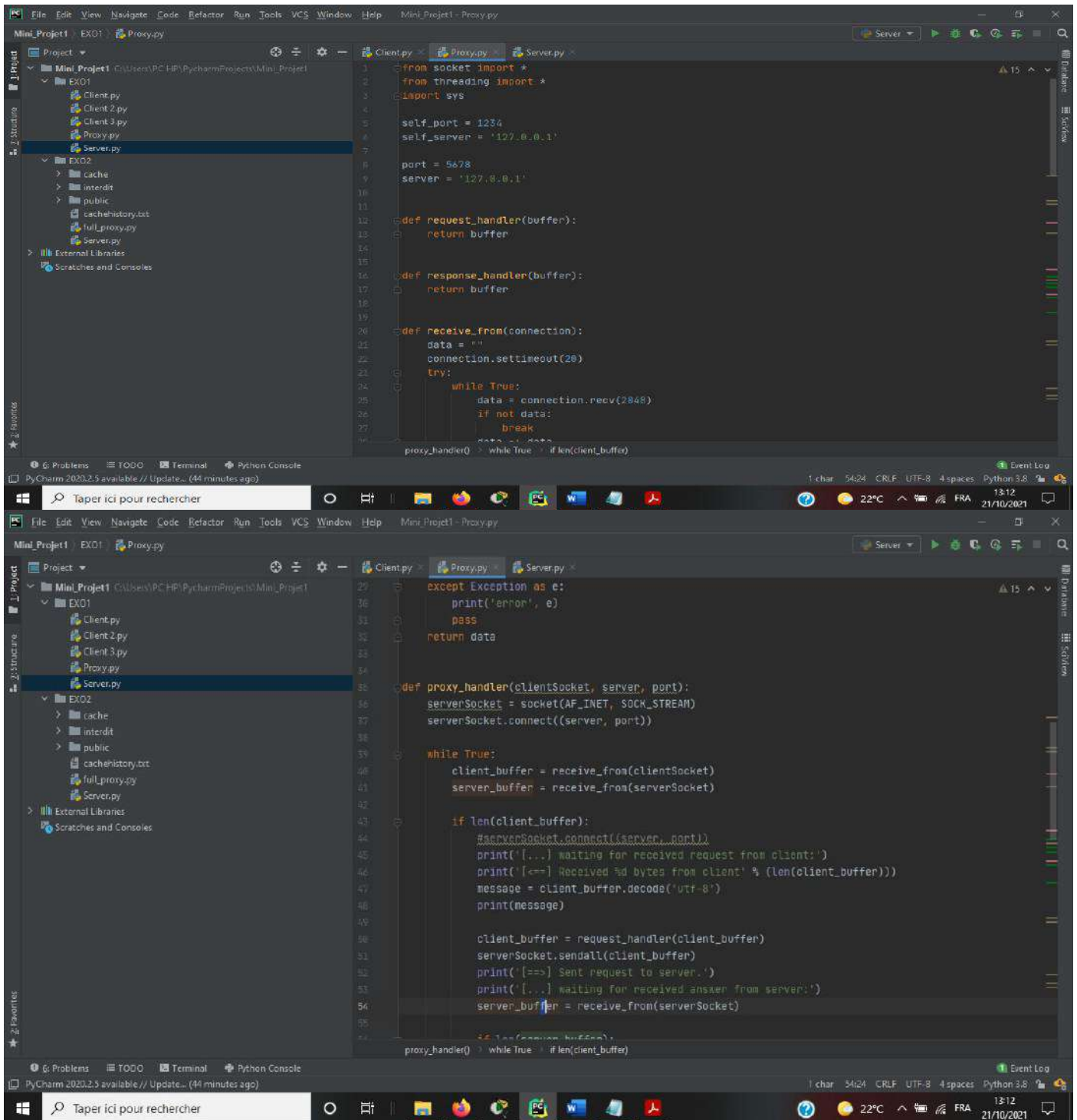
La partie du client a des simples tâches à faire il connecte d'abord au buffer pour transmettre sa requête il envoie la requête il reste à l'écoute pour recevoir sa réponse.

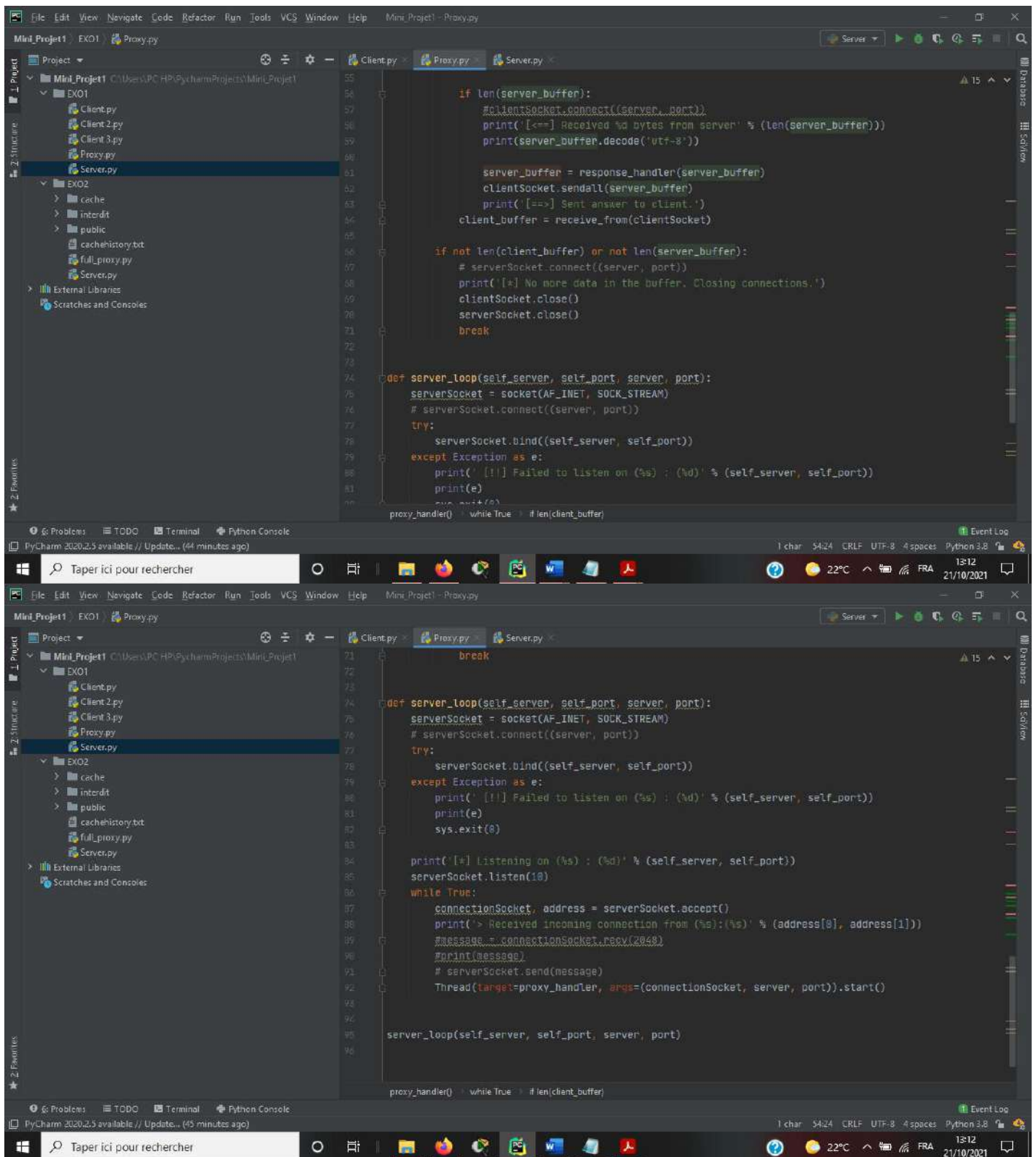


```
1 from socket import *
2
3 proxy = '127.0.0.1'
4 port = 1234
5
6 clientSocket = socket(AF_INET, SOCK_STREAM)
7 clientSocket.connect((proxy, port))
8
9 message = bytearray([15] * 5)
10 print('[==>] sent request to proxy.')
11 clientSocket.send(message)
12
13 print('[...] Waiting for receive answer from server.')
14 response = clientSocket.recv(2048)
15 print('[<==] Receive answer from proxy.')
16 print(response.decode())
17
18 print('[*] Nothing to send. Closing connection')
19 clientSocket.close()
20
```

2.1.2 Proxy :

Le coté qui va faire plus de tâche d'abord il écoute sur le buffer pour capte aucune requête envoyée alors si le client qui a envoyé une requête (request) il fait l'acheminement vers le serveur pour lui envoyer la requête demandée par le client. Il attend toujours le traitement de la requête par le serveur et lorsqu'il reçoit la réponse il fait l'opération inverse il envoie la réponse au client.





2.1.3 Serveur :

Le serveur sa tâche et aussi simple pareil au client lorsqu'il reçoit une requête de la part du client il a traité (encodage – décodage) ensuite il envoie une réponse ou bien un acquittement au proxy pour l'envoyer au client.

```
1 from socket import *
2 from threading import *
3
4 port = 5678
5 server = '127.0.0.1'
6
7 client_port = 1234
8 client_server = '127.0.0.1'
9
10 serverSocket = socket(AF_INET, SOCK_STREAM)
11 serverSocket.bind((server, port))
12 serverSocket.listen(5)
13
14 print('[*] Server listening on (%s) : (%d)' % (server, port))
15
16
17 def handle_client(clientSocket):
18     print('[...] waiting for received data from proxy:')
19     received = clientSocket.recv(2048)
20     print('[<==] Received %d bytes from proxy' % (len(received)))
21     print(received.decode())
22     clientSocket.sendall(b'ACK')
23     #clientSocket.sendto(b'ACK', (client_server, client_port))
24     print('[==>] Sent answer to proxy.')
25
26
27 while True:
28     connectionSocket, address = serverSocket.accept()
29     handle_client()
```

```
14 print('[*] Server listening on (%s) : (%d)' % (server, port))
15
16
17 def handle_client(clientSocket):
18     print('[...] waiting for received data from proxy:')
19     received = clientSocket.recv(2048)
20     print('[<==] Received %d bytes from proxy' % (len(received)))
21     print(received.decode())
22     clientSocket.sendall(b'ACK')
23     #clientSocket.sendto(b'ACK', (client_server, client_port))
24     print('[==>] Sent answer to proxy.')
25
26
27 while True:
28     connectionSocket, address = serverSocket.accept()
29     print('[*] Accepted connection from (%s):(%s)' % (address[0], address[1]))
30     #message = connectionSocket.recv(2048)
31     #print(message)
32     #serverSocket.sendto(message, ('127.0.0.1', 1234))
33     Thread(target=handle_client, args=(connectionSocket,)).start()
34
35 serverSocket.close()
36
37 handle_client()
```

2.1.4 Plusieurs clients :

Lorsqu'on veut avoir plusieurs clients qui envoient leurs requêtes au proxy ou serveur là on a besoin d'introduire le mécanisme des threads pour gérer plusieurs clients à la fois. Par contre le principe et le code de chaque client reste le même.

3 Exercice 2 Proxy HTTP :

On cherche à programmer en Python un proxy dédié au protocole HTTP. On considère donc que tous les échanges entre le client et le serveur (via le proxy) utilisent le protocole HTTP. Le client est donc typiquement un navigateur web, et le serveur est donc typiquement un serveur web.

3.1 Explication des décisions techniques :

3.1.1 Client :

Un navigateur web qui va saisir sa demande GET

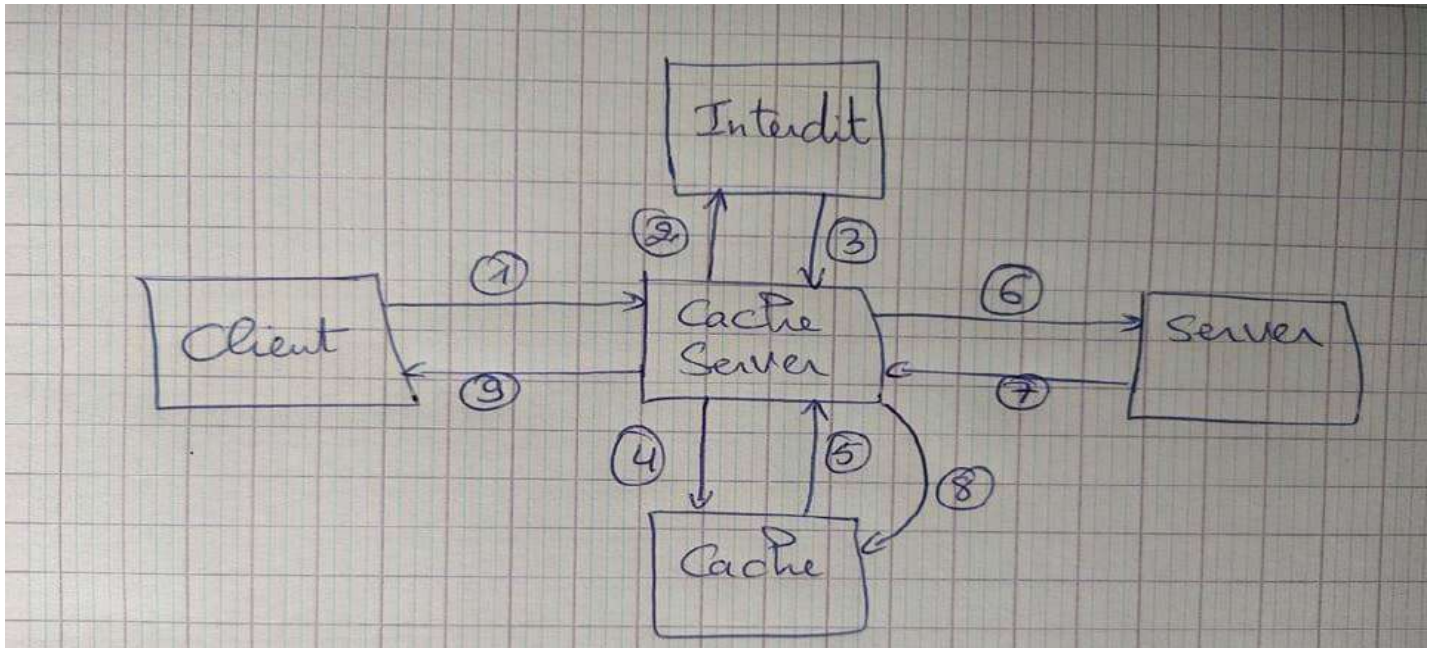
3.1.2 Proxy :

d'abord il écoute sur le buffer pour capte aucune requête envoyée il prend la requête il teste d'abord si le get demandé ou bien le fichier existe sur la liste interdit si oui il renvoie un message 403 Forbidden sinon il a 2 cas : soit le fichier demandé existe dans le cache dans il transmet le contenu directement au client message 200 OK sinon il doit envoyer la requête au serveur pour lui demander le fichier si il a eu une réponse positif du serveur il stock d'abord le fichier dans son cache et il envoie le contenu du fichier ou bien la réponse au client s'il a eu une réponse négative du serveur il envoie un message d'erreur 404 NOT FOUND au client que le fichier n'existe pas.

3.1.3 Serveur :

Sa tâche est facile il reçoit la requête de la part du proxy il la traite si le fichier existe il l'envoie au proxy sinon il envoie fichier non trouvé.

3.2 Voici un schéma d'illustration du principe de fonctionnement :



- 1- Le client fait une requête.
- 2- Le cache vérifie si fichier demandé existe dans son dossier des fichiers censurés.
- 3- Le fichier n'existe pas s'il existe le cache envoie au client 403 Forbidden (fichier interdit).
- 4- Le cache cherche si le fichier existe dans son dossier local cache.
- 5- Le fichier n'existe pas s'il existe le cache envoie son contenu directement au client 200 OK.
- 6- Le cache demande au serveur de lui envoyer le fichier demandé par le client.
- 7- Le serveur envoie une réponse avec succès ou bien réponse négative (fichier non trouvé).
- 8- Le cache stock une copie du fichier reçu par le serveur si réponse positive du serveur.
- 9- Ensuite le cache envoie la réponse ou le contenu du fichier au client sinon une réponse d'erreur 404 NOT FOUND (fichier non trouvé).

4 Conclusion

Ce mini-projet a été une belle expérience à titre éducative et professionnelle pour comprendre d'abord les différentes étapes de communication entre un client et serveur lorsqu'un client fait une requête que ce soit par un navigateur web ou bien sur un terminal. Et le rôle d'un serveur cache entre ces deux terminaux.

Il nous a permis aussi de comprendre les avantages d'un serveur proxy que ce soit pour la réduction de charge sur le serveur ou pour la rapidité de transmettre la réponse au client et aussi son mécanisme de sécurité d'interdire certains fichiers à passer au client.

Et enfin on a su comment établir et implémenter tout ce mécanisme en réalité.