

## TP N° 05 - Module M2207

Le but de ce TP est de se familiariser avec la programmation d'un serveur de transfert de fichiers texte en Python. Une sorte d'équivalent de FTP...

### 1. Client pour serveur de transfert de fichiers :

Le protocole que nous allons utiliser est le suivant :



Le client affichera la chaîne **PROMPT** (rt207-2 dans l'exemple ci-dessous) envoyée par le serveur, et capturera en boucle un nom de fichier (**FILE\_NAME**). Il enverra alors ce nom au serveur et attendra le nombre de lignes à venir **NB\_LIGNES**, puis il lira ces NB lignes.

Si le fichier n'existe pas, le serveur enverra 1 puis un message d'erreur sur une seule ligne.

Si le fichier est trop gros (plus de 10 lignes), le serveur enverra 1 puis un message d'erreur sur une seule ligne.

Voici un exemple d'utilisation :

```
python3 txtFileClient.py
rt207-2$ users.txt
<= 3
<= toto:titi
<= tata:tutu
<= titi:titi
rt207-2$ hello
<= 1
<= Erreur de fichier 'hello': [Errno 2] No such file or directory!
rt207-2$ txtFileClient.py
<= 1
<= Fichier [txtFileClient.py] trop gros !
rt207-2$ @bye
<= 1
<= Au revoir !
```

- a) Créer le script `txtFileClient.py` pour qu'il suive le protocole ci-dessus.

On pourra utiliser l'exécutable fourni sur moodle (`fileServer-x64-linux`) ou bien l'image docker disponible sur le « registry » de l'IUT pour les tests :

```
docker pull registry.iutbeziers.fr/cb_file:1.0
docker run --rm -t --name cb registry.iutbeziers.fr/cb_file:1.0
...# A partir de là, lancer les clients dans une autre console vers l'adresse de l'interface docker0+1 (e.g. 172.17.0.2)
...# when finished
docker stop cb
```

### 2. Serveur de commandes :

- Créer le script `txtFileServer.py` qui indique dans un premier temps, toujours une erreur sur le fichier demandé.
- Déterminer ensuite le nombre de lignes du fichier demandé en faisant par exemple une lecture complète du fichier avec la méthode `read()`, puis en comptant le nombre de sauts de ligne dans le résultat de la lecture (voir méthode `count(substr)`).
- Modifier ensuite le client et le serveur (port 9877) pour chiffrer les données avec RSA (Voir TD n° 04).  
On n'enverra plus le fichier ligne par ligne, mais bloc par bloc (avec des chaînes hexadécimales, voir ci-dessous).  
Le serveur utilisera une clé RSA privée pour chiffrer et le client la clé RSA publique correspondante pour déchiffrer.  
On pourra donc créer la méthode `sendstrCipher()` sur le serveur et la fonction `readstrDecipher()` sur le client.

```
python3 txtFileClientRSA.py
Client TXT File RSA
<= pccb.borelly.net
pccb.borelly.net$ txtFileClientRSA.py
=> txtFileClientRSA.py
readstrDecipher()
<= 11
<=
0x1e71defef92065c612501eba7ce945936d787a80dea9b4e37e3f1c74e14beba6ed7b16c5038de15
7f572c6391932f45f87aad7ec865b8bf8dc4138d16879a37654f992bf49e20863af73e6bdf87a286b
43121fcc6363ba6aa615c551e518a71ceea9f85ffd87cfc9c0bf2380101f4c639bd9a1a1f9096de88
8d46285bb47d678
<=
0x5b33400061c951ec14670f6a970a7b62b0a86504d80b4e3fc1c714b4eea2064f447351301ad7d91
1633b038ef631ce35dd69fa57f9dbc7f7ba4aca8eaa6959d89bf624add23889299134e01d969a25f92
b021a18506c81ccb4bf90afe757d08b06dbe8cbc1c74873db6d84232b4eade3156737720550f01b77
c758512e9459cef
...
<== #!/usr/bin/env python3
# -*- coding: utf-8 -*-
import socket,sys
...
pccb.borelly.net$ @bye
=> @bye
readstrDecipher()
<= 1
<=
0x21a29558afe3ed97d784dffb7534e93374e02ac430c0f779f4788641390a8077283219d56b3e1e3
b7c146fa2c1e3c5df630d8eef456e1fe310f1a2f1fd51318550aa8ffdd7c3577920b7a238fdceba14
673eb4e29d9f5255eb8cc972bf34fc4a01470b476a819ac69cc68ab7ac918c60d7d3688858fcd49e7
e2a32e9391262e1
<== Au revoir !
Close
```