Python et Unix : TP2

Préambule :

L'objectif de ce TP est de mettre en place un modèle client/serveur à l'aide des API socket. Réutilisez au maximum les interactions déjà mises en place au TP1.

Exercice 1:

Nous allons implémenter un protocole client serveur, permettant d'accéder à des fichiers placés coté Serveur. Ce protocole permet les trames de requêtes suivantes :

101 PASSWD string
102 ACCES filename
103 QUIT

Les réponses sont composées des trames suivantes :

901 OK
902 INCONNU
903 NOPERM
904 NONTROUVE
905 EOF

Le protocole se déroule de la manière suivante :

- A l'initialisation de la connexion le serveur envoie une trame OK
- Le client doit avant tout s'authentifier à l'aide d'une trame PASSWD avec son mot de passe : si ce dernier est correct on renvoie une trame OK, sinon une trame INCONNU
- Toute requête du client s'il ne s'est pas authentifié donne une trame NOPERM
- Le client peut demander à accéder un fichier : trame ACCES avec le nom du fichier
    - Le serveur cherche le fichier dans le répertoire local :
        - s'il n'existe pas, on renvoie NONTROUVE
        - s'il existe on renvoie OK, suivi des lignes du fichier, suivi de EOF, le client les affiche simplement
- Le client peut terminer la session à l'aide de QUIT (on lui réponds OK avant de fermer la connexion)

Indications complémentaires :

- Utilisez des sockets TCP (cf. support)
- Dans un fichier protocole.py, créez deux listes *requetes* et *reponses* contenant les requêtes et les réponses. Pour émettre des trames ou reconnaître une trame reçue comparez à *protocole.requetes[i]* ou *protocole.reponses[i]* afin d'assurer la cohérence du client et du serveur.
- Pour reconnaître une trame et la découper pour trouver par exemple le mot de passe, utilisez des slices sur les chaines reçues
- Utilisez un port libre > 1024

- Pour tester la présence d'un fichier utilisez le code suivant :

```
try :
        f = open (path_to_fich)
except :
        print "fichier non trouve"
```

Si le fichier à été trouvé f.readlines() vous rends une liste de toutes les lignes du fichier, f.readline() permet de lire le fichier ligne par ligne (cf. documentation)

Exercice 2

[1]Votre serveur n'est actuellement capable de traiter qu'un seul client à la fois. Nous allons en faire un serveur multi-thread. A l'aide de *os.fork()* vu dans les supports ou du modèle suivant, implémentez le multi-thread coté serveur.

Le fork permet de créer un nouveau processus, les mécanismes de thread sont plus légers et indépendants de l'os. Le framework d'un serveur multi-thread peut suivre ce schéma :

```
import thread

def f(x1,x2):
   # code de traitement d'un client

code_principal    # ici seul le processus initial est lancé

            # un nouveau client arrive ...
thread.start_new_thread(f,arg1,arg2)
# Lance un nouveau Thread qui execute la fonction f avec arg1 et arg2
# Il se termine quand f se termine

ici suite du code pour le thread principal
#seul le processus initial arrive à cet endroit
```

Pour tester la cohérence de votre application, connectez vous avec votre client au serveur de votre voisin.

[1] Ce sujet est librement inspiré du support proposé par l'université sciences cognitives de Sussex (GB) en intiation à python.