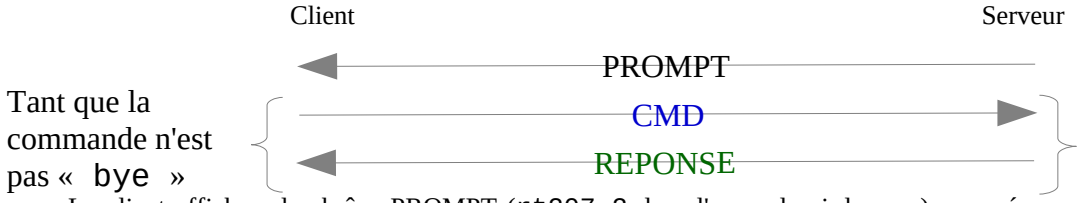
TP N° 04 - Module M2207

Le but de ce TP est de se familiariser avec la programmation d'un serveur de « commandes » en Python. Une sorte d'équivalent de Telnet/SSH...

1. Client pour serveur de commandes de base :

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 une commande au clavier (CMD). Il enverra alors la commande au serveur et affichera la REPONSE (sur une ligne dans un premier temps) tant que la commande n'est pas égale à « `bye` ».

Voici un exemple d'utilisation :

```
python3 cmdClient.py
rt207-2$ hello
Commande [hello] inconnue !
rt207-2$ world
Commande [world] inconnue !
rt207-2$ bye
Au revoir !
```

a) Modifier le client (créer le script `cmdClient.py`) pour qu'il suive le protocole ci-dessus. Utiliser le port TCP 5678 pour les tests.

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

```
docker pull registry.iutbeziers.fr/cb_cmd:1.0
docker run --rm -t --name cb registry.iutbeziers.fr/cb_cmd: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 :

a) Adapter ensuite le code du serveur de base (Créer le script `cmdServer.py`).

b) On veut ensuite envoyer le nom de la machine comme prompt. Utiliser alors la méthode `gethostname()` du module `socket`.

c) Pour exécuter une commande sur le serveur et obtenir le résultat, on peut utiliser le module `subprocess`.

```
try:
  res=subprocess.run(['ls'],stdout=subprocess.PIPE,stderr=subprocess.PIPE)
  print('[%s]'%(res.stdout))
except OSError as err:
  print('Erreur:',err,file=sys.stderr)
```

Tester le code sur le serveur avec les commandes `id`, `pwd` ou `uname`.

d) Quand le résultat tient sur plusieurs lignes, on ne peut pas facilement envoyer les données au client car le client ne connaît pas le nombre de ligne qu'il doit lire. Une solution consiste à encoder les données en base64 :

```
>>> import base64
>>> mbytes=b'abcd'
>>> b64=base64.b64encode(mbytes)
>>> print(b64)
b'YWJjZA=='
>>> deco=base64.b64decode(b64)
>>> print(deco)
b'abcd'
```

Adapter votre code sur le client et le serveur.

e) Comment limiter les commandes possibles sur le serveur à la liste `['id','pwd','uname','ls']` ?

f) Que se passe-t-il quand on utilise des arguments dans une commande ? Comment résoudre le problème ? Voir peut être le module `shlex` par exemple...

g) Comment obtenir le message d'erreur si la commande renvoie un code d'erreur non nul (voir attribut `returncode`) ?

h) Le système est-il sécurisé ? Tester par exemple les commandes suivantes dans un terminal classique et sur le client CMD :

```
ls ; cat /etc/passwd
ls && cat /etc/passwd
```