

TP5a - Socket Client

1 Introduction aux sockets avec le protocole Daytime

1.1 Utilisation du service Daytime avec Telnet

Telnet est un protocole simple de connexion à distance : il permet de transmettre des caractères entre une machine locale (écran + clavier) et une machine distante¹.

Par défaut, un client telnet se connecte au service TCP *telnet* (sur le port 23), mais il est possible de préciser le service TCP de la façon suivante : `telnet M S`. Dans ce cas, le client *telnet* se connecte au service S de la machine M.

- Quel est l'effet de la commande suivante ? `telnet time.nist.gov daytime`
- Décrire le protocole *daytime* d'après le résultat de cette commande. Vous trouverez sur le site web de l'IETF (*Internet Engineering Task Force*) la spécification officielle de ce protocole, décrite dans la [RFC 867](#).
- Il est également possible d'indiquer explicitement le numéro de port du service plutôt que le nom du service. Consultez le fichier `/etc/services` et cherchez avec `grep` le numéro de port du service *daytime*. Essayez maintenant de vous connecter avec le numéro de port trouvé.
- Avec la commande de terminal `grep`, retrouvez efficacement les numéros de port par défaut associés aux services (ou protocoles) suivant : http, ftp, smtp, telnet, ssh, echo.
- Au CREMI, la machine *quirrell* accueille le service *daytime* en version TCP et UDP sur IPv4 et IPv6. A l'aide de la commande `nc quirrell 13`, essayez de vous connecter aux 4 différentes versions de ce service : on utilisera les options `-u`, `-4`, `-6`. Consultez le *manuel en ligne* de cette commande pour comprendre le rôle des options : `man nc` (faire 'q' pour quitter le manuel). Pourquoi ne peut-on pas utiliser la commande *telnet* pour faire ces expériences ? Rappelez les principales différences entre TCP et UDP. Expliquez rapidement la différence de fonctionnement du service *daytime* entre TCP et UDP ?

Pour info pour plus tard : Pour forcer la terminaison d'une session Telnet, il faut faire `Ctrl+Alt+Gr+]`, puis il faut taper `quit` lorsque l'invite `telnet>` s'affiche.

1.2 Programmation d'un client Daytime avec les Socket

Considérons le programme en Python 3 ci-dessous, qui permet de se connecter au service *daytime* à l'aide de la bibliothèque *socket*, disponible sur moodle : [daytime.py](#)

1. À essayer depuis chez vous : `telnet towel.blinkenlights.nl`.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "time-c.nist.gov"
port = 13
s.connect((host, port))
data = s.recv(1024)
print(data)
s.close()
```

- Ouvrez votre éditeur de texte (ou environnement de programmation) préféré et recopiez le code de ce programme dans un fichier *daytime.py*.
- Lancez ce programme à l'aide de la commande : `python3 daytime.py`
- A l'aide de la documentation (<https://docs.python.org/library/socket.html>), comprenez les étapes principales de ce programme et répondez aux questions suivantes. En particulier, expliquez le sens des constantes `AF_INET` et `SOCK_STREAM`. Quel est le rôle de la variable *s*? Quelles lignes déclenchent la connexion & la déconnexion TCP/IP? Quelle fonction sert à recevoir des données dans ce code? Cherchez dans la documentation la fonction qui sert à envoyer des données?
- Pour transformer ce programme en script exécutable, il faut commencer par ajouter sur la première ligne : `#!/usr/bin/python3`, puis il faut donner les droits d'exécution (+x) au fichier *daytime.py* en tapant la commande en ligne suivante : `chmod +x daytime.py`. Vous pouvez maintenant lancer ce programme directement : `./daytime.py`

2 Socket Python : requête HTTP à la main

2.1 Echauffons-nous d'abord avec Telnet

Nous avons déjà étudié avec *Wireshark* ce qui se passe quand un navigateur consulte une page web, comme par exemple <http://www.perdu.com>. Revenons rapidement sur la trace capturée dans le fichier [http.pcap](#). A l'aide de Wireshark, observez en détail dans l'en-tête HTTP la requête GET du client (trame 10).

A l'aide de la commande *telnet*, reproduisez la requête "HTTP GET" vers le serveur www.perdu.com en se limitant aux options *Host* et *Connection* :

```
$ telnet www.perdu.com 80
GET / HTTP/1.1
Host: perdu.com
Connection: close
```

Attention, il ne faut pas oublier le double saut de ligne à la fin de la requête! Aussi, le serveur n'est pas très patient et produit rapidement une erreur **Timeout**, il vaut mieux préparer le texte prêt à coller depuis un éditeur de texte.

2.2 Programmons notre client web !

Écrivons maintenant dans le fichier *httpget.py* un programme Python 3 qui récupère la page d'accueil d'un site web (passé en argument), dont l'utilisation sera :

```
$ ./httpget.py www.perdu.com
```

Nous allons nous inspirer de *daytime.py* écrit précédemment. Puisque l'on veut ouvrir une connexion TCP sur le port 80, il faut adapter de la façon suivante :

- Importer le module *sys* avec `import sys`, afin de récupérer l'argument `sys.argv[1]` dans la variable `host`.
- Mettre à jour la variable `port`.
- Ajoutez une variable `request` qui contient la requête HTTP (disponible sur moodle : [http-request.py](#))

```
request = "GET / HTTP/1.1\r\n" \
          "Host: " + host + "\r\n" \
          "Connection: close\r\n\r\n"
```

- Utiliser la méthode `sendall()` de l'objet socket pour envoyer la requête (contrairement au protocole daytime, ici, c'est le client qui commence à parler!). Attention, il faut convertir la chaîne de caractères `request` en tableau d'octets avant de l'envoyer : `request.encode("utf-8")` qui retourne un tableau d'octets. Pensez bien à ajouter deux retours chariot `\r\n\r\n` à la fin de la requête.
- Utiliser la méthode `recv()` de l'objet socket pour récupérer le résultat (on peut lui passer la taille 1024). Il suffit ensuite d'utiliser un `print()` pour l'afficher.
- Testez votre programme avec le site web [www.gnu.org](#). Pourquoi ce dernier est-il tronqué ?
- Que votre requête soit correcte ou pas, le serveur va toujours retourner une page web du moment que la connexion est faite. Comment savoir si le serveur a bien compris votre requête ? Corrigez éventuellement votre code.

2.3 Raffinements

Pour récupérer toute la page, il faut mettre une boucle autour des appels `recv()` et `print()`. Lorsque le serveur referme la connexion TCP parce que la page est finie, `recv()` retourne le tableau d'octets vide `b""`, il suffira alors de tester cela pour savoir quand s'arrêter.

Par ailleurs, il faut noter que `recv()` retourne un tableau d'octets qu'il faut convertir en chaîne de caractères (*str*) Python avec la fonction `decode()` avant de l'afficher avec `print()`. Comme la chaîne de caractères contient déjà des retours lignes `"\n"`, il n'est pas nécessaire que la fonction `print()` en rajoute (option `end=""`) :

```
answer = s.recv(1024)
# affichage en byte array
print(answer)
# decodage et affichage en string utf-8
print(answer.decode("utf-8"), end="")
```

En cas d'erreur lors des étapes de connexion, il vaut mieux afficher un gentil message à l'utilisateur plutôt que laisser l'exception terminer le programme ! On peut récupérer une exception en entourant l'opération de socket avec la construction syntaxique suivante :

```
try:
    s.truc()
except Exception as e:
    print("socket error!", e)
    sys.exit(1)
```

Traitez différemment les cas d'erreur pour apporter à l'utilisateur une aide spécifique à chaque cas : essayez par exemple avec `localhost` (sur lequel aucun serveur web ne tourne) et avec `trucbidule` (qui n'existe pas), cela lève une exception différente.