

Applications client-serveur

Nom & Prénom : Arij Mabrouk

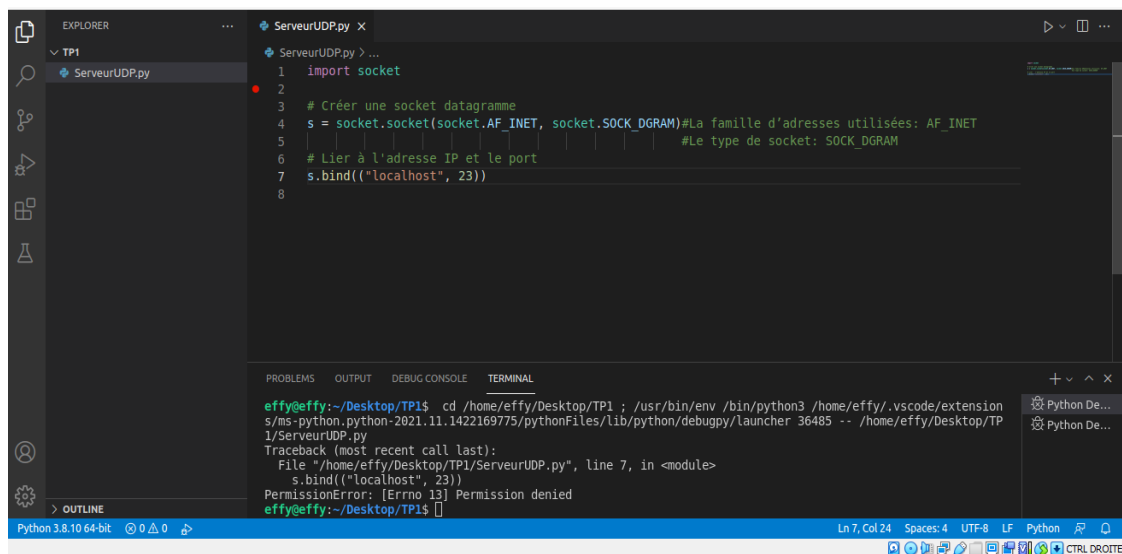
Nom & Prénom : Jouhaina Nasri

Groupe : 2 IDL 1

1 Programmation client-serveur

Exercice 1.1 : Programmation client-serveur

1. Écrivez un serveur UDP qui ouvre une socket et l'associe au port 23.



The screenshot shows a VS Code editor with a file named `ServeurUDP.py` open. The code in the editor is as follows:

```
1 import socket
2
3 # Créer une socket datagramme
4 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # La famille d'adresses utilisées: AF_INET
5 # Le type de socket: SOCK_DGRAM
6 # Lier à l'adresse IP et le port
7 s.bind(("localhost", 23))
8
```

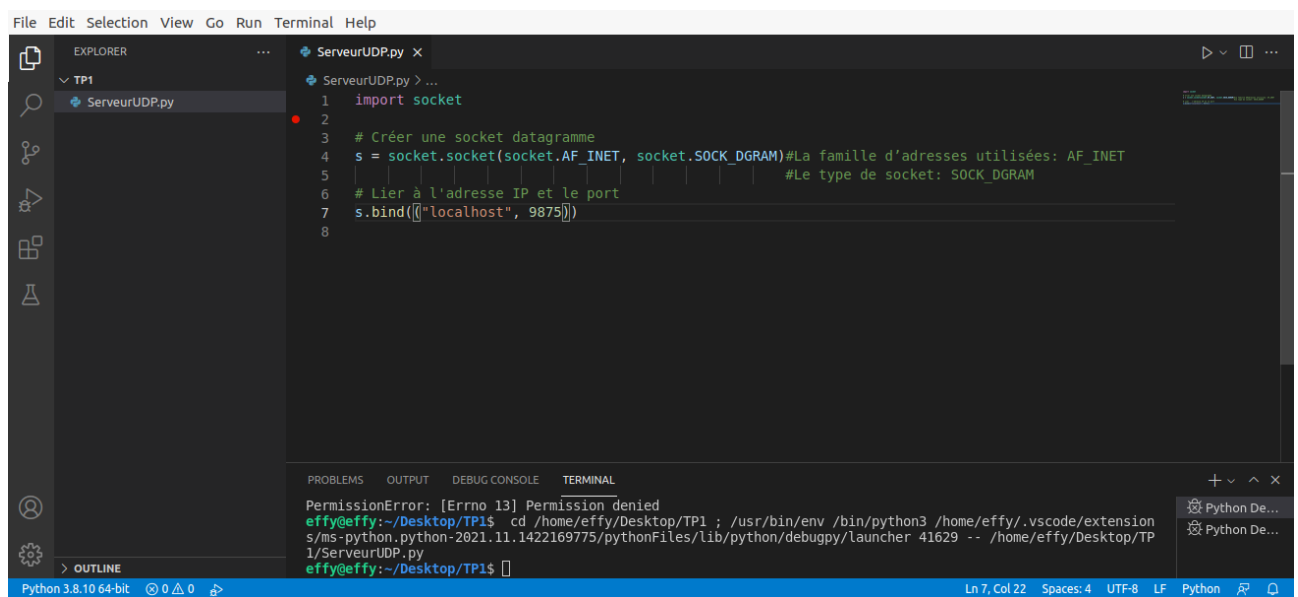
Below the editor, the TERMINAL panel shows the command executed and the resulting error:

```
effy@effy:~/Desktop/TP1$ cd /home/effy/Desktop/TP1 ; /usr/bin/env /bin/python3 /home/effy/.vscode/extension
s/ms-python.python-2021.11.1422169775/pythonFiles/lib/python/debugpy/launcher 36485 -- /home/effy/Desktop/TP
1/ServeurUDP.py
Traceback (most recent call last):
  File "/home/effy/Desktop/TP1/ServeurUDP.py", line 7, in <module>
    s.bind(("localhost", 23))
PermissionError: [Errno 13] Permission denied
effy@effy:~/Desktop/TP1$
```

2. Lancez votre serveur sur une autre machine de la salle TP. Que se passe-t-il?

Le serveur va afficher un message d'erreur dû à l'utilisation restreinte du port 23 car les ports inférieurs à 1024 sont réservés au super-utilisateur : une application exécutée par un utilisateur normal ne pourra pas ouvrir ces ports, qui sont pour la plupart réservés à des services connus

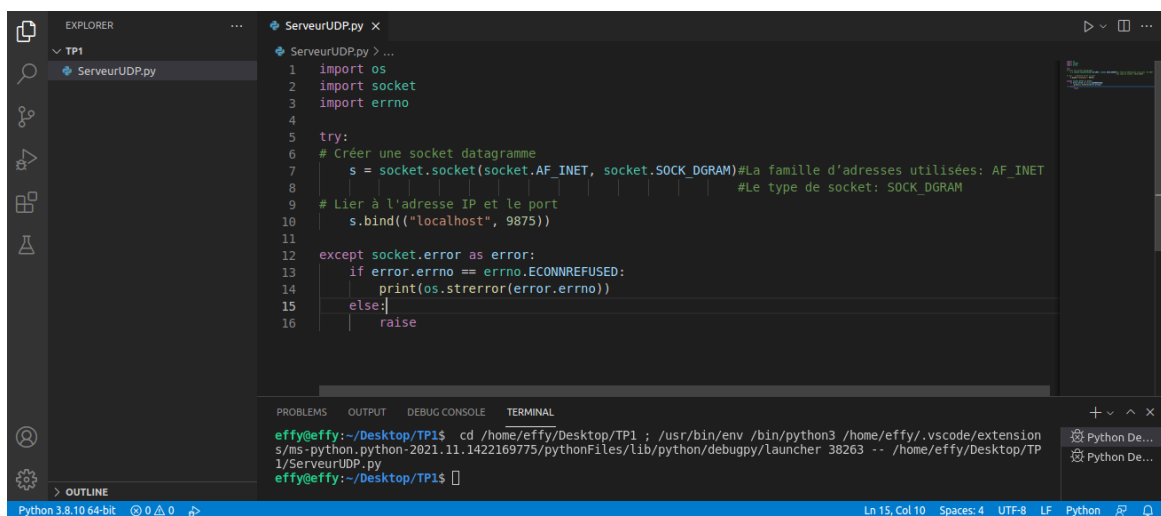
3. Modifiez votre serveur pour associer votre socket au port 9875. Que se passe-t-il maintenant quand vous lancez votre serveur ?



```
File Edit Selection View Go Run Terminal Help
EXPLORER
TP1
  ServeurUDP.py
  ...
  ServeurUDP.py x
    1 import socket
    2
    3 # Créer une socket datagramme
    4 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #La famille d'adresses utilisées: AF_INET
    5                                           #Le type de socket: SOCK_DGRAM
    6 # Lier à l'adresse IP et le port
    7 s.bind(("localhost", 9875))
    8

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PermissionError: [Errno 13] Permission denied
effy@effy:~/Desktop/TP1$ cd /home/effy/Desktop/TP1 ; /usr/bin/env /bin/python3 /home/effy/.vscode/extension
s/ms-python.python-2021.11.1422169775/pythonFiles/lib/python/debugpy/launcher 41629 -- /home/effy/Desktop/TP
1/ServeurUDP.py
effy@effy:~/Desktop/TP1$
```

4. Modifiez votre programme afin d'afficher un message d'erreur et quitter proprement le programme dans le cas où une exception est levée.



```
EXPLORER
TP1
  ServeurUDP.py
  ...
  ServeurUDP.py x
    1 import os
    2 import socket
    3 import errno
    4
    5 try:
    6     # Créer une socket datagramme
    7     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #La famille d'adresses utilisées: AF_INET
    8                                           #Le type de socket: SOCK_DGRAM
    9     # Lier à l'adresse IP et le port
    10    s.bind(("localhost", 9875))
    11
    12 except socket.error as error:
    13     if error.errno == errno.ECONNREFUSED:
    14         print(os.strerror(error.errno))
    15     else:
    16         raise

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
effy@effy:~/Desktop/TP1$ cd /home/effy/Desktop/TP1 ; /usr/bin/env /bin/python3 /home/effy/.vscode/extension
s/ms-python.python-2021.11.1422169775/pythonFiles/lib/python/debugpy/launcher 38263 -- /home/effy/Desktop/TP
1/ServeurUDP.py
effy@effy:~/Desktop/TP1$
```

5. Modifiez votre serveur afin qu'il puisse recevoir un message. Si une exception est soulevée, n'oubliez pas de fermer la socket avec la fonction close() avant de quitter le programme.

```
1 import os
2 import socket
3 import errno
4
5 try:
6     # Créer une socket datagramme
7     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #La famille d'adresses utilisées: AF_INET
8                                                         #Le type de socket: SOCK_DGRAM
9     # Lier à l'adresse IP et le port
10    s.bind(("localhost", 9875))
11
12    #question5: fonction recvfrom()
13    data,adr = s.recvfrom(1024) #taille du tampon=1024
14                                #donnée reçu= data
15                                #adresse du client= adr
16
17
18
19
20 except socket.error as error:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1/ServerUDP.py
effy@effy:~/Desktop/TP1\$ cd /home/effy/Desktop/TP1 ; /usr/bin/env /bin/python3 /home/effy/.vscode/extension s/ms-python.python-2021.11.1422169775/pythonFiles/lib/python/debugpy/launcher 33787 -- /home/effy/Desktop/TP 1/ServerUDP.py

Python 3.8.10 64-bit

6. (Un serveur écoute en permanence ce qui vient. Modifiez votre programme pour mettre le `recvfrom()` dans une boucle infinie. Lorsqu'un message est reçu, affichez le message reçu et l'adresse source.

```
4
5
6 # Créer une socket datagramme
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #La famille d'adresses utilisées: AF_INET
8                                                         #Le type de socket: SOCK_DGRAM
9 # Lier à l'adresse IP et le port
10 s.bind(("localhost", 9875))
11 while True :
12     #question5: fonction recvfrom()
13     data,adr = s.recvfrom(1024) #taille du tampon=1024
14                                 #donnée reçu= data
15                                 #adresse du client= adr
16
17     clientMsg = "Message du client: {}".format(data)
18     clientIP = "Adresse IP du client: {}".format(adr)
19     print([clientMsg])
20     print(clientIP)
21
22 except socket.error as error:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

KeyboardInterrupt
effy@effy:~/Desktop/TP1\$ cd /home/effy/Desktop/TP1 ; /usr/bin/env /bin/python3 /home/effy/.vscode/extension s/ms-python.python-2021.11.1422169775/pythonFiles/lib/python/debugpy/launcher 43799 -- /home/effy/Desktop/TP 1/ServerUDP.py

Python 3.8.10 64-bit

7. Lancez votre serveur. Normalement, votre programme ne devrait pas vous rendre la main : il est dans le `recvfrom()`. Vous pouvez voir les ports utilisés sur votre système avec l'utilitaire `netstat`. Ce programme prend quelques options en paramètres, parmi lesquels les protocoles concernés, les

informations visualisées... Regardez l'aide en ligne de netstat et affichez les ports UDP utilisés sur votre machine. Normalement vous devriez voir votre serveur écouter sur le port que vous avez choisi.

```
effy@effy: ~  
effy@effy:~$ netstat -lu  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State  
udp      0      0 0.0.0.0:mdns             0.0.0.0:*  
udp      0      0 localhost:domain         0.0.0.0:*  
udp      0      0 0.0.0.0:631              0.0.0.0:*  
udp      0      0 0.0.0.0:50820            0.0.0.0:*  
udp      0      0 localhost:9875           0.0.0.0:*  
udp6     0      0 [::]:mdns                [::]:*  
udp6     0      0 [::]:59911               [::]:*
```

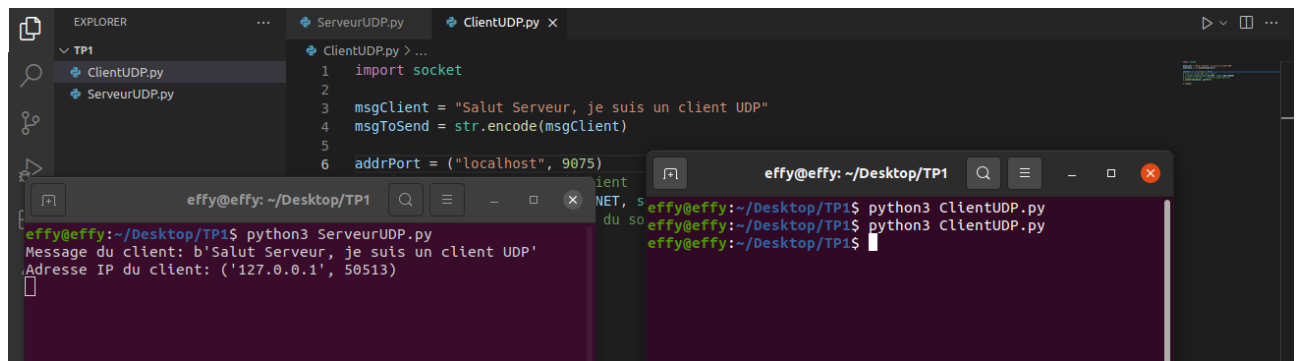
3.1.1 Client UDP

8. Écrivez un programme client qui crée une socket UDP et envoie un message sous forme d'une chaîne de caractères à votre serveur. Côté serveur, le programme doit afficher le message reçu.

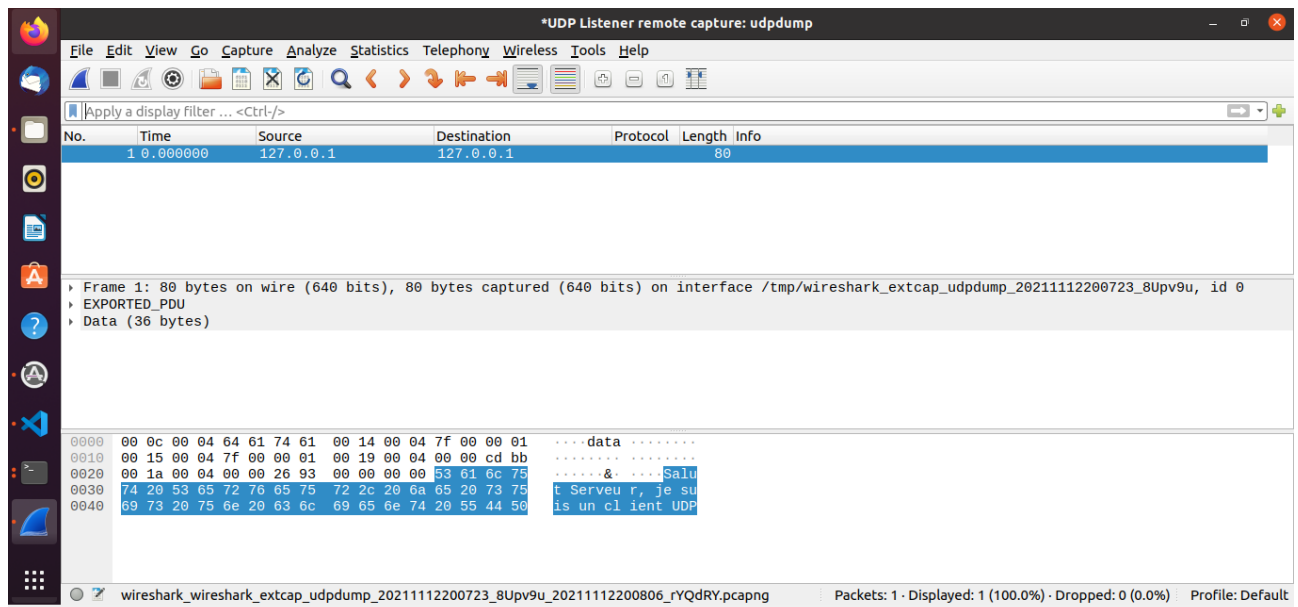
```
EXPLORER  
v TP1  
+ ClientUDP.py  
+ ServeurUDP.py  
+ ClientUDP.py x  
+ ClientUDP.py > ...  
1 import socket  
2  
3 msgClient = "Salut Serveur, je suis un client UDP"  
4 msgToSend = str.encode(msgClient)  
5  
6 addrPort = ("localhost", 9875)  
7 # Créer un socket UDP coté client  
8 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
9 # Envoyer au serveur à l'aide du socket UDP créé  
10 s.sendto(msgToSend, addrPort)  
11  
12 s.close()
```

```
effy@effy: ~/Desktop/TP1  
effy@effy:~/Desktop/TP1$ python3 ServeurUDP.py  
Message du client: b'Salut Serveur, je suis un client UDP'  
Adresse IP du client: ('127.0.0.1', 50513)  
  
effy@effy: ~/Desktop/TP1  
effy@effy:~/Desktop/TP1$ python3 ClientUDP.py  
un cl  
  
cket.  
ket U  
  
py
```

9. Si le port n'est pas correct, que se passe-t-il ?



10. Avec l'outil Wireshark vous pouvez capturer le trafic passant sur une interface réseau. Lancez Wireshark sur votre machine et lancez une capture, exécutez votre client et arrêtez la capture. Quels sont les messages effectivement échangés sur le réseau ?



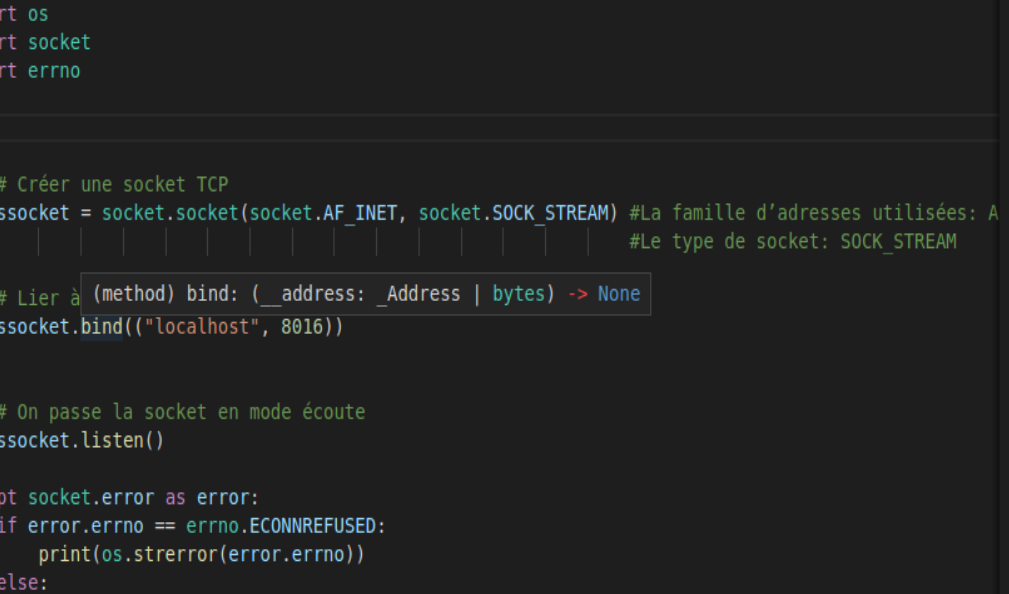
1.1 Client et serveur TCP

3.2.1 Serveur TCP

11. Écrivez un programme serveur qui ouvre une socket TCP et l'associe au port de votre choix.

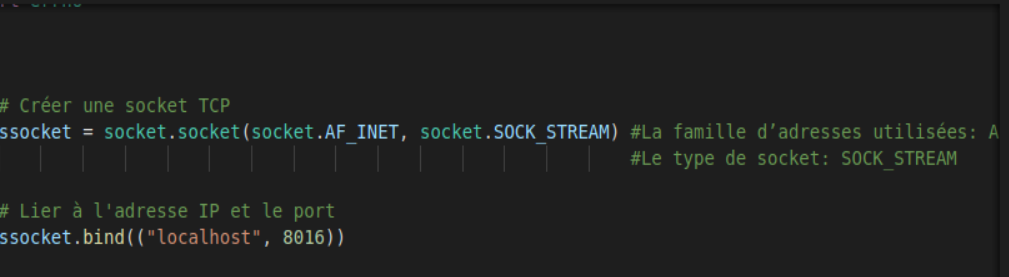
[illegible]

12. Modifiez votre programme serveur pour passer votre socket en mode écoute.



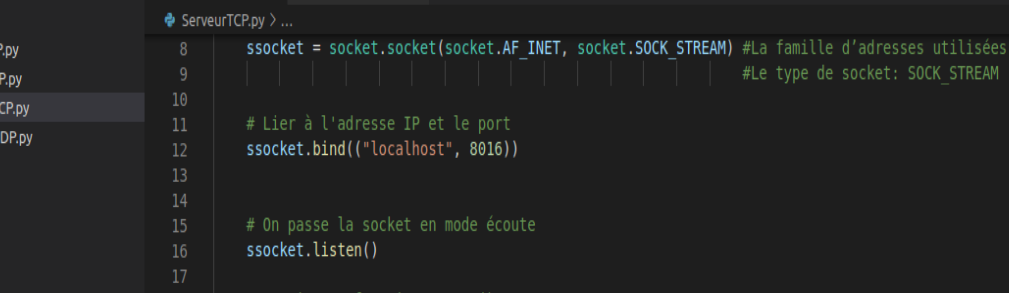
```
1 import os
2 import socket
3 import errno
4
5
6 try:
7     # Créer une socket TCP
8     ssocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #La famille d'adresses utilisées: A
9                                     #Le type de socket: SOCK_STREAM
10
11     # Lier à (method) bind: (_address: _Address | bytes) -> None
12     ssocket.bind(("localhost", 8016))
13
14
15     # On passe la socket en mode écoute
16     ssocket.listen()
17
18 except socket.error as error:
19     if error.errno == errno.ECONNREFUSED:
20         print(os.strerror(error.errno))
21     else:
22         raise
23
24 ssocket.close()
```

13. Modifiez votre programme serveur pour accepter les connexions entrantes et afficher l'adresse du client qui vient de se connecter.



```
1 import socket
2
3
4
5
6 try:
7     # Créer une socket TCP
8     ssocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #La famille d'adresses utilisées: AF_INET
9                                     #Le type de socket: SOCK_STREAM
10
11     # Lier à l'adresse IP et le port
12     ssocket.bind(("localhost", 8016))
13
14
15     # On passe la socket en mode écoute
16     ssocket.listen()
17
18 #question13: fonction accept()
19 while True:
20     conn, addr = ssocket.accept()
21
22
23 except socket.error as error:
24     if error.errno == errno.ECONNREFUSED:
25         print(os.strerror(error.errno))
26     else:
27         raise
28
```

14. Modifiez votre serveur pour recevoir des données du client.



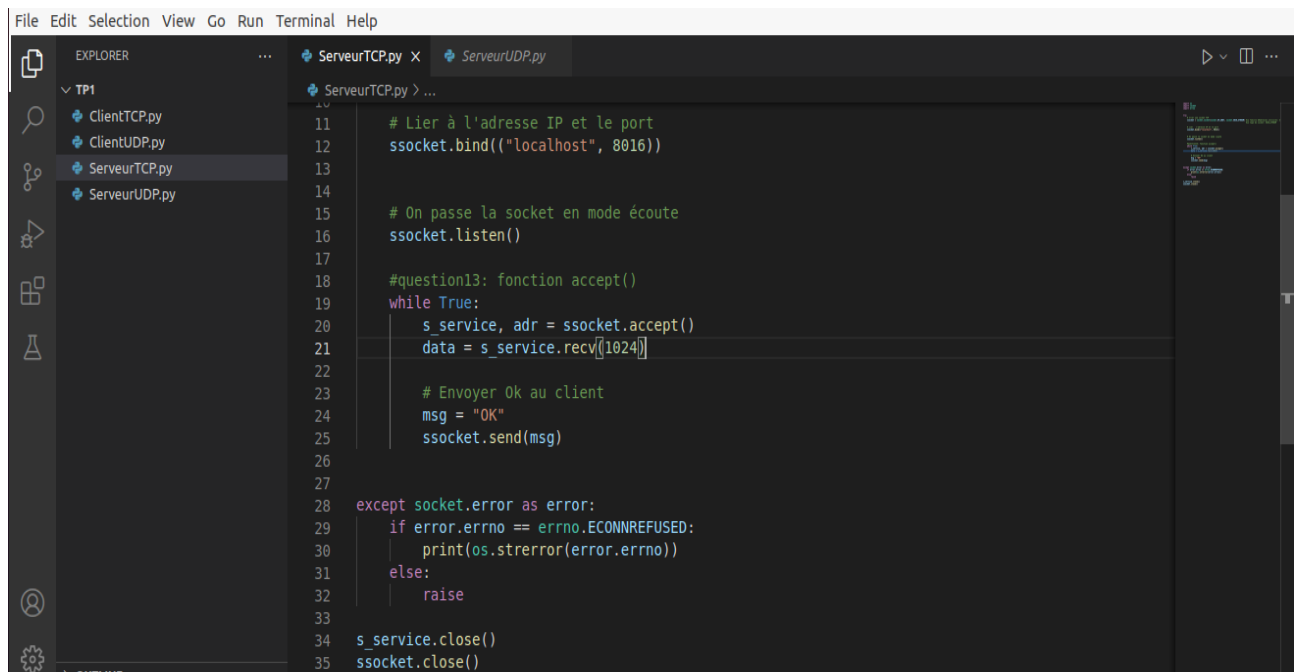
```
File Edit Selection View Go Run Terminal Help

EXPLORER
TP1
  ClientTCP.py
  ClientUDP.py
  ServeurTCP.py
  ServeurUDP.py

ServeurTCP.py > ...

8      ssocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #La famille d'adresses utilisées: AF_INET
9      #Le type de socket: SOCK_STREAM
10
11     # Lier à l'adresse IP et le port
12     ssocket.bind(("localhost", 8016))
13
14
15     # On passe la socket en mode écoute
16     ssocket.listen()
17
18     #question13: fonction accept()
19     while True:
20         s_service, adr = ssocket.accept()
21         data = s_service.recv(1024)
22
23
24
25     except socket.error as error:
26         if error.errno == errno.ECONNREFUSED:
27             print(os.strerror(error.errno))
28         else:
29             raise
30
31     s_service.close()
32     ssocket.close()
```

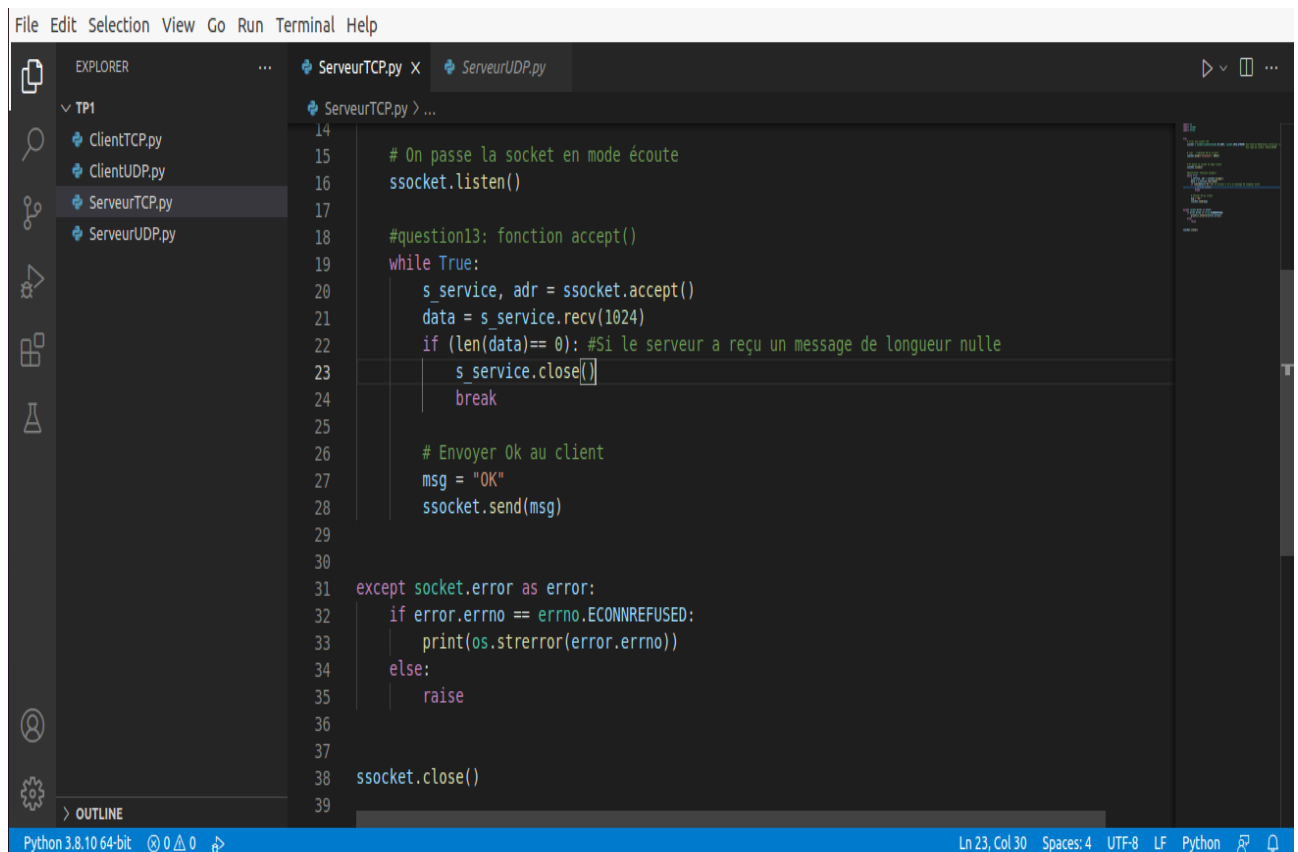

15. Modifiez votre serveur pour qu'il envoie ce message au client.



The screenshot shows the Visual Studio Code editor with the file `ServeurTCP.py` open. The code is as follows:

```
11 # Lier à l'adresse IP et le port
12 ssocket.bind(("localhost", 8016))
13
14
15 # On passe la socket en mode écoute
16 ssocket.listen()
17
18 #question13: fonction accept()
19 while True:
20     s_service, adr = ssocket.accept()
21     data = s_service.recv(1024)
22
23     # Envoyer Ok au client
24     msg = "OK"
25     ssocket.send(msg)
26
27
28 except socket.error as error:
29     if error.errno == errno.ECONNREFUSED:
30         print(os.strerror(error.errno))
31     else:
32         raise
33
34 s_service.close()
35 ssocket.close()
```

16. Modifiez votre programme pour que le serveur détecte la fermeture de connexion côté client et ferme sa socket de communication avec ce client.

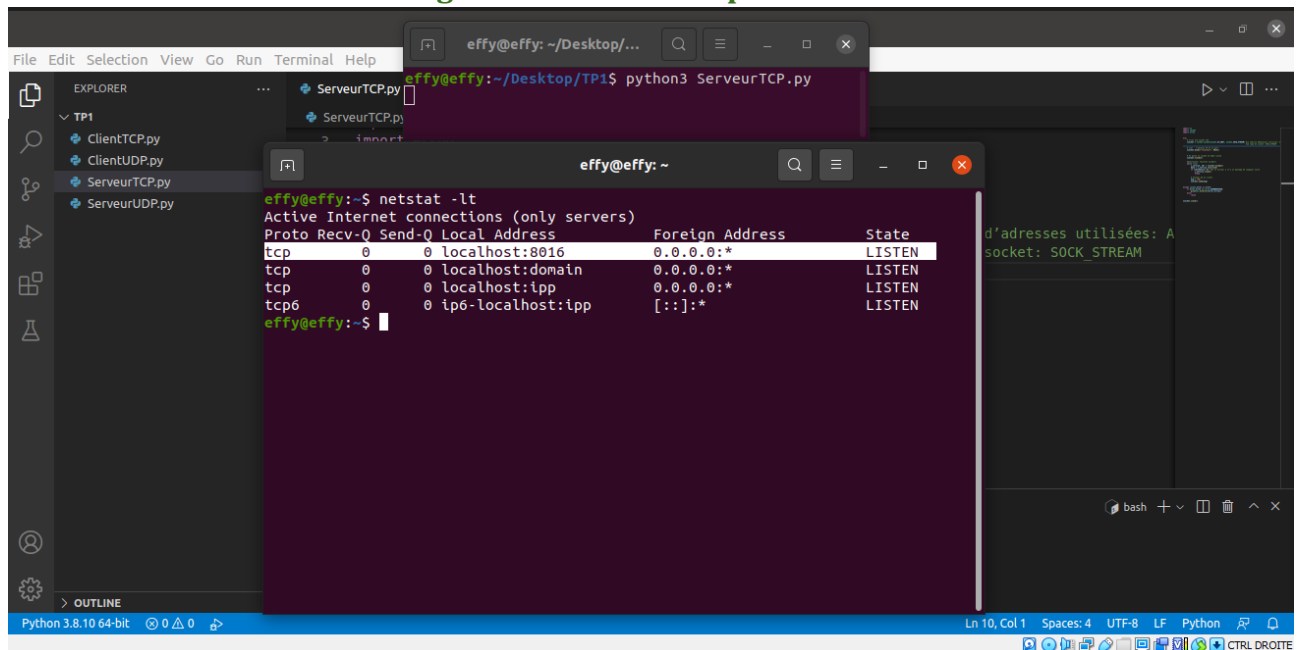


The screenshot shows the Visual Studio Code editor with the file `ServeurTCP.py` open. The code is as follows:

```
14
15 # On passe la socket en mode écoute
16 ssocket.listen()
17
18 #question13: fonction accept()
19 while True:
20     s_service, adr = ssocket.accept()
21     data = s_service.recv(1024)
22     if (len(data)== 0): #Si le serveur a reçu un message de longueur nulle
23         s_service.close()
24         break
25
26     # Envoyer Ok au client
27     msg = "OK"
28     ssocket.send(msg)
29
30
31 except socket.error as error:
32     if error.errno == errno.ECONNREFUSED:
33         print(os.strerror(error.errno))
34     else:
35         raise
36
37
38 ssocket.close()
39
```

The status bar at the bottom indicates: Python 3.8.10 64-bit, 0 errors, 0 warnings, 0 info, Ln 23, Col 30, Spaces: 4, UTF-8, LF, Python.

17. Lancez votre serveur et regardez l'état de son port avec netstat.

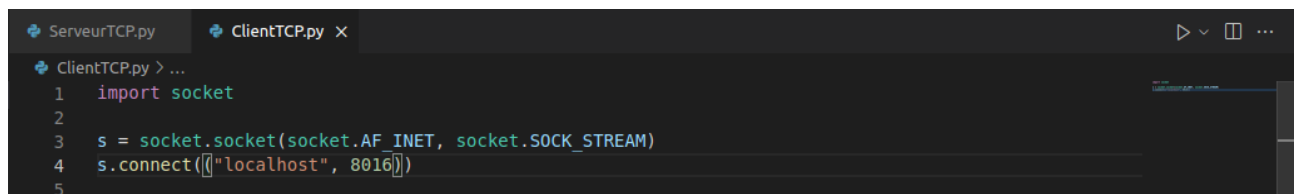


```
effy@effy: ~/Desktop/TP1$ python3 ServeurTCP.py

effy@effy: ~$ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address State
tcp        0      0 localhost:8016   0.0.0.0:*       LISTEN
tcp        0      0 localhost:domain 0.0.0.0:*       LISTEN
tcp        0      0 localhost:ipp    0.0.0.0:*       LISTEN
tcp6       0      0 ip6-localhost:ipp [::]:*         LISTEN
effy@effy:~$
```

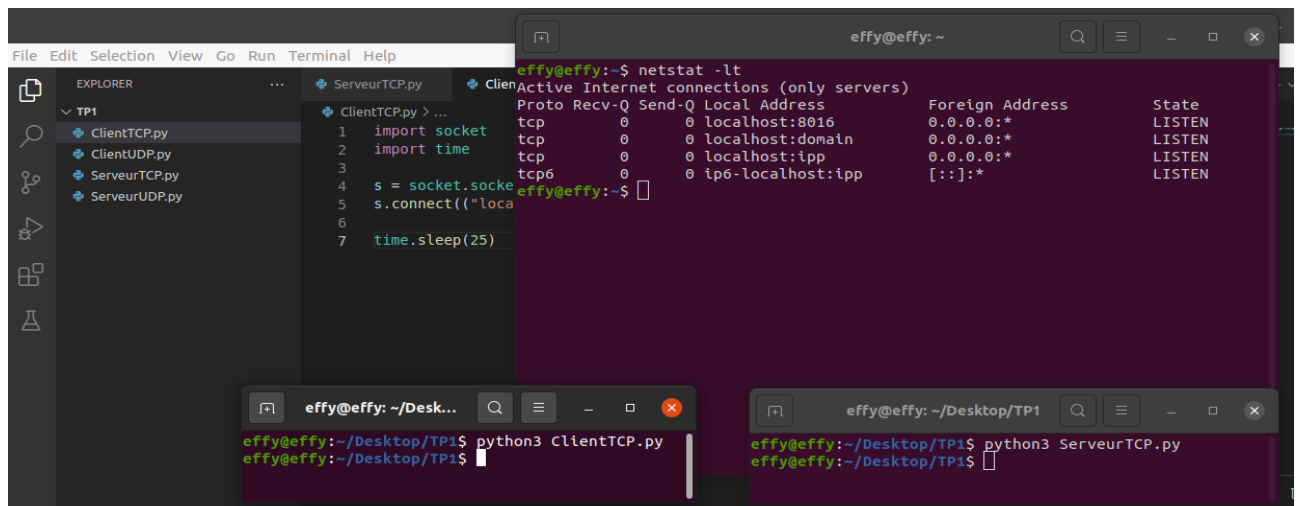
3.2.2 Client TCP

18. Écrivez un programme client qui se connecte à votre serveur TCP.



```
ClientTCP.py > ...
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 s.connect(("localhost", 8016))
5
```

19. Lancez maintenant netstat sur la machine sur laquelle s'exécute le serveur et constatez l'état des connexions ouvertes avec le serveur.



```
effy@effy: ~$ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address State
tcp        0      0 localhost:8016   0.0.0.0:*       LISTEN
tcp        0      0 localhost:domain 0.0.0.0:*       LISTEN
tcp        0      0 localhost:ipp    0.0.0.0:*       LISTEN
tcp6       0      0 ip6-localhost:ipp [::]:*         LISTEN
effy@effy:~$

effy@effy:~/Desktop/TP1$ python3 ClientTCP.py

effy@effy:~/Desktop/TP1$ python3 ServeurTCP.py
```

20. Modifiez votre programme client pour envoyer et recevoir un message, puis fermer la connexion avec le serveur.

```

ClientTCP.py
15
16 #récupérer le nom du machine pour l'envoyer au serveur
17 host=socket.gethostname()
18 msgToSend = str.encode(host)
19
20
21 #le client a envoyé son nom de machine au serveur
22 s.send(msgToSend)
23
24
25 #le client a reçu un msg
26 msg = s.recv(1024)
27
28 #Afficher le message reçu du serveur
29 serveurMsg = "Message du serveur: {}".format(msg)
30 print(serveurMsg)
31
32
33
34
35
36 except socket.error as error:
37     if error.errno == errno.ECONNREFUSED:
38         print(os.strerror(error.errno))
39     else:
40         raise

ServeurTCP.py
25
26 # Envoyer Ok au client
27 msg = str.encode("OK")
28 s_service.send(msg)
29
30 #Afficher le msg reçu du client
31 clientMsg = "Message du client: {}".format(data)
32 print(clientMsg)
33
34 except socket.error as error:
35     if error.errno == errno.ECONNREFUSED:
36         print(os.strerror(error.errno))
37     else:
38         raise
39
40 ssocket.close()
41
42
  
```

```

effy@effy: ~/Desktop/TP1
effy@effy:~/Desktop/TP1$ python3 ServeurTCP.py
Message du client: b'effy'

effy@effy:~/Desktop/TP1$ python3 ClientTCP.py
Message du serveur: b'OK'
effy@effy:~/Desktop/TP1$
  
```

21. Lancez une capture avec Wireshark pour capturer l'intégralité des échanges entre votre client et votre serveur. Quels messages sont échangés effectivement sur le réseau

No.	Time	Source	Destination	Protocol	Length	Info
5	15.233651551	127.0.0.1	127.0.0.1	TCP	68	44178 → 8016 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=6878613
6	15.233787312	127.0.0.1	127.0.0.1	TCP	72	44178 → 8016 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=68
7	15.233792873	127.0.0.1	127.0.0.1	TCP	68	8016 → 44178 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=6878613
8	15.233810611	127.0.0.1	127.0.0.1	TCP	70	8016 → 44178 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=2 TSval=68
9	15.233854204	127.0.0.1	127.0.0.1	TCP	68	44178 → 8016 [ACK] Seq=5 Ack=3 Win=65536 Len=0 TSval=6878613
10	15.233882823	127.0.0.1	127.0.0.1	TCP	68	44178 → 8016 [FIN, ACK] Seq=5 Ack=3 Win=65536 Len=0 TSval=68
11	15.287217305	127.0.0.1	127.0.0.1	TCP	68	8016 → 44178 [ACK] Seq=3 Ack=6 Win=65536 Len=0 TSval=6878613

Frame 10: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface any, id 0
 Linux cooked capture
 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 44178, Dst Port: 8016, Seq: 5, Ack: 3, Len: 0

```

0000  00 00 03 04 00 06 00 00 00 00 00 00 00 08 00  .....@.
0010  45 00 00 34 a0 eb 40 00 40 06 9b d6 7f 00 00 01  E..@. @.
0020  7f 00 00 01 ac 92 1f 50 6c 02 40 4a 1f 5e 28 5d  ....P 1.@.^[
0030  80 11 02 00 fe 28 00 00 01 01 08 0a 28 ff ee 53  .....(..S
0040  28 ff ee 53  .....(..S
  
```

2 Finger

1 Les services sous Unix

1.1 Ports des services réseaux

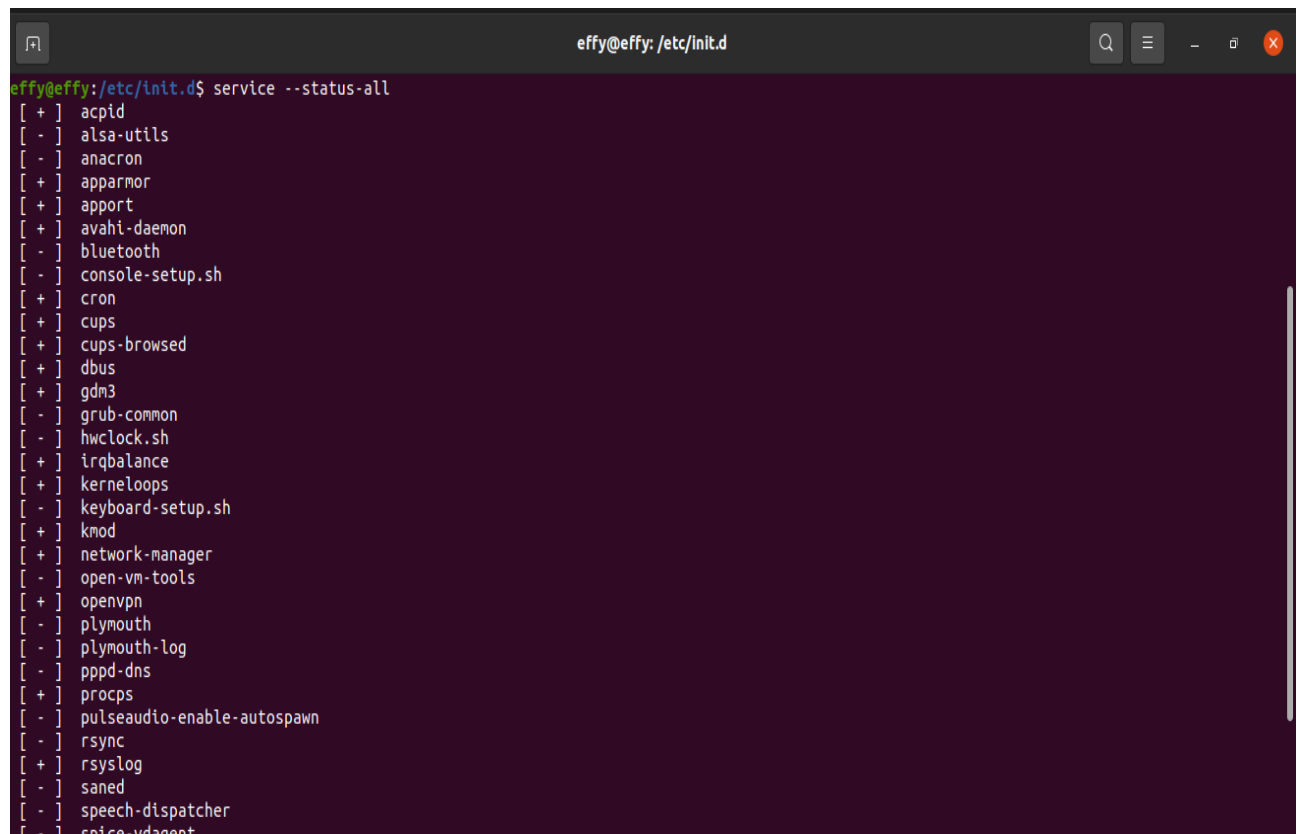
1. Quels sont le protocole et le numéro de port associés à finger ?

Le numéro de port associé au finger est 79

Le protocole est TCP

1.2 Scripts de démarrage

2. Quels sont les services disponibles sur vos machines ?



```
effy@effy:/etc/init.d$ service --status-all
[ + ] acpid
[ - ] alsa-utils
[ - ] anacron
[ + ] apparmor
[ + ] apport
[ + ] avahi-daemon
[ - ] bluetooth
[ - ] console-setup.sh
[ + ] cron
[ + ] cups
[ + ] cups-browsed
[ + ] dbus
[ + ] gdm3
[ - ] grub-common
[ - ] hwclock.sh
[ + ] irqbalance
[ + ] kerneloops
[ - ] keyboard-setup.sh
[ + ] kmod
[ + ] network-manager
[ - ] open-vm-tools
[ + ] openvpn
[ - ] plymouth
[ - ] plymouth-log
[ - ] pppd-dns
[ + ] procps
[ - ] pulseaudio-enable-autospawn
[ - ] rsync
[ + ] rsyslog
[ - ] saned
[ - ] speech-dispatcher
[ + ] spice-vdagent
```

```
effy@effy: /etc/init.d
[ - ] console-setup.sh
[ + ] cron
[ + ] cups
[ + ] cups-browsed
[ + ] dbus
[ + ] gdm3
[ - ] grub-common
[ - ] hwclock.sh
[ + ] irqbalance
[ + ] kerneloops
[ - ] keyboard-setup.sh
[ + ] kmod
[ + ] network-manager
[ - ] open-vm-tools
[ + ] openvpn
[ - ] plymouth
[ - ] plymouth-log
[ - ] pppd-dns
[ + ] procps
[ - ] pulseaudio-enable-autospawn
[ - ] rsync
[ + ] rsyslog
[ - ] saned
[ - ] speech-dispatcher
[ - ] spice-vdagent
[ + ] udev
[ + ] ufw
[ + ] unattended-upgrades
[ - ] uuid
[ + ] whoopsie
[ - ] x11-common
effy@effy:/etc/init.d$
```

3. Comment les utilise-t-on pour lancer, arrêter ou redémarrer un service ?

- Pour lancer un service : nom_de_service **enable**
- Pour arrêter un service: nom_de_service **disable**
- Pour redémarrer un service: nom_de_service **reset**

Et ce-dessous un exemple sur le service ufw:

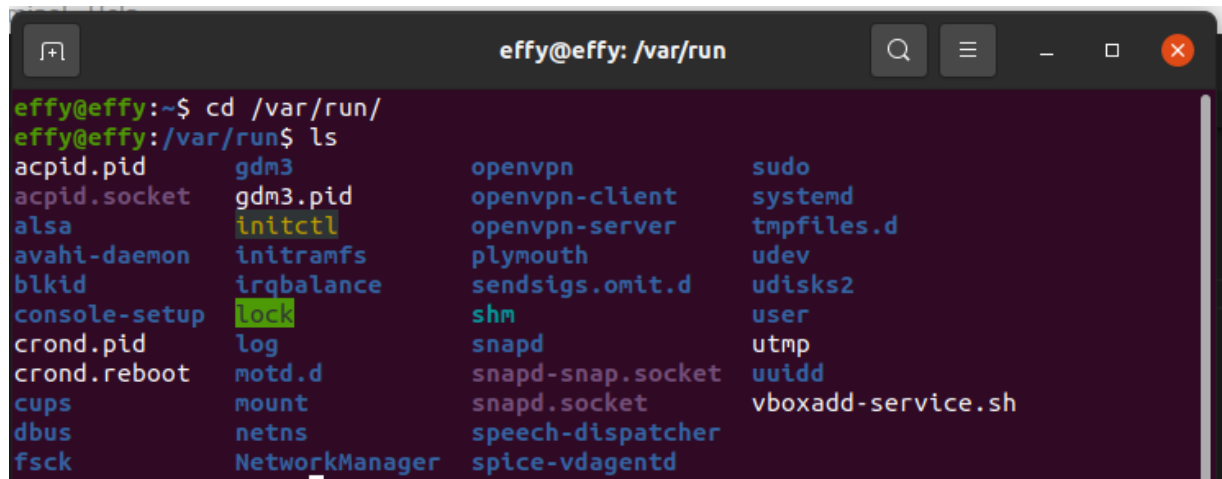
```
effy@effy: /etc/init.d
effy@effy:/etc/init.d$ sudo ufw reset
Resetting all rules to installed defaults. Proceed with operation (y|n)? y
Backing up 'user.rules' to '/etc/ufw/user.rules.20211122_221930'
Backing up 'before.rules' to '/etc/ufw/before.rules.20211122_221930'
Backing up 'after.rules' to '/etc/ufw/after.rules.20211122_221930'
Backing up 'user6.rules' to '/etc/ufw/user6.rules.20211122_221930'
Backing up 'before6.rules' to '/etc/ufw/before6.rules.20211122_221930'
Backing up 'after6.rules' to '/etc/ufw/after6.rules.20211122_221930'

effy@effy:/etc/init.d$ sudo ufw disable
Firewall stopped and disabled on system startup
effy@effy:/etc/init.d$ sudo ufw enable
Firewall is active and enabled on system startup
effy@effy:/etc/init.d$ sudo ufw status
Status: active
effy@effy:/etc/init.d$
```

1.3 Fichiers de traces et pid

4. Regardez les fichiers contenus dans ce répertoire. Lesquels contiennent le pid des services ? Comment est enregistré le pid dans ces fichiers ?

le pid des services sont enregistrés dans des fichiers isolés comme l'illustre la capture ci-dessous:

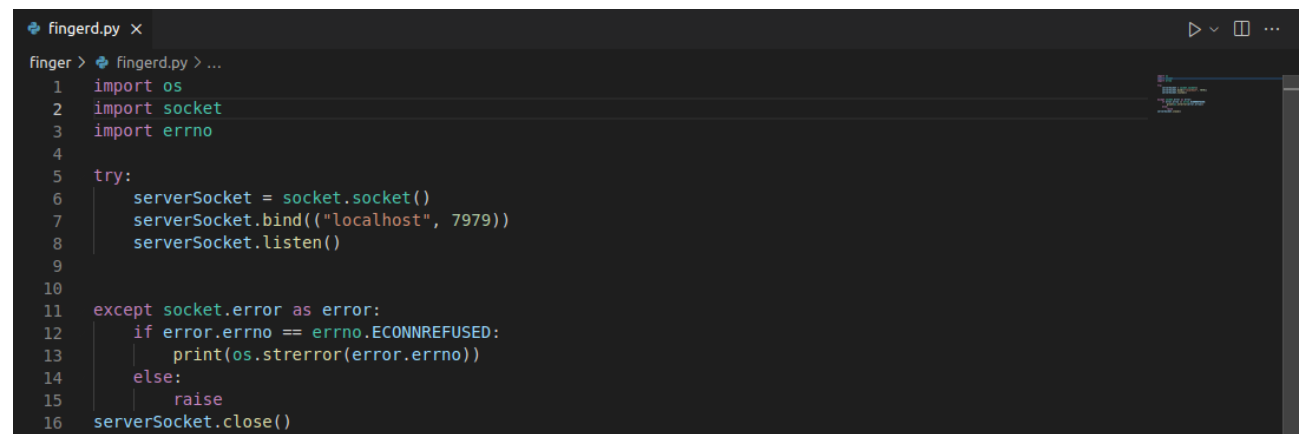


```
effy@effy: /var/run
effy@effy:~$ cd /var/run/
effy@effy:/var/run$ ls
acpid.pid      gdm3          openvpn        sudo
acpid.socket   gdm3.pid      openvpn-client systemd
alsa           initctl       openvpn-server tmpfiles.d
avahi-daemon   initramfs     plymouth       udev
blkid          irqbalance    sendsigs.omit.d udisks2
console-setup  lock          shm            user
crond.pid      log           snapd          utmp
crond.reboot   motd.d        snapd-snap.socket uuid
cups           mount         snapd.socket   vboxadd-service.sh
dbus           netns         speech-dispatcher
fsck           NetworkManager spice-vdagentd
```

5. Les services écrivent des informations sur ce qui se passe durant leur exécution dans des fichiers de traces. Ces fichiers se trouvent dans le répertoire /var/log. On peut journaliser les erreurs qui surviennent (log d'erreurs) ou tous types d'évènements (logs d'accès par exemple). Regardez le contenu de quelques fichiers de logs pour voir ce qu'ils contiennent.

2 Implémentation du protocole Finger

6. Écrivez un programme Python fingerd.py serveur qui écoute sur le port TCP 7979.



```
fingerd.py x
Finger > fingerd.py > ...
1 import os
2 import socket
3 import errno
4
5 try:
6     serverSocket = socket.socket()
7     serverSocket.bind(("localhost", 7979))
8     serverSocket.listen()
9
10
11 except socket.error as error:
12     if error.errno == errno.ECONNREFUSED:
13         print(os.strerror(error.errno))
14     else:
15         raise
16 serverSocket.close()
```

7. En vous aidant de la documentation Python en ligne, modifiez votre serveur pour exécuter cette commande lors de la réception d'une chaîne de caractères et récupérer sa sortie.

```
fingerd.py X
finger > fingerd.py > ...
1 import os
2 import socket
3 import subprocess
4 import errno
5
6 try:
7     serverSocket = socket.socket()
8     serverSocket.bind(("localhost", 7979))
9     serverSocket.listen()
10
11 #fonction accepte qui accepte tous les cnx et retourne socket du client et son adresse
12 (client, adrr) = serverSocket.accept()
13 while (True):
14     login = client.recv(1024)#récupérer login
15     print(login)#affichage login
16     concat = b"finger" + b' ' + login #concaténation du "finger" avec la chaîne de caractères reçus(login)
17
18     if login != b'':#si login <> du vide
19
20         out = subprocess.getoutput(concat)#le module commands n'exist plus en python3
21         #il est donc remplacé par celui du subprocess
22
23     else:
24         serverSocket.close()
25         break
26
27 except socket.error as error:
28     if error.errno == errno.ECONNREFUSED:
29         print(os.strerror(error.errno))
30     else:
31         raise
32 serverSocket.close()
```

8. Modifiez votre programme serveur pour envoyer la sortie de la commande finger au client.

```
fingerd.py X
finger > fingerd.py > ...
9 serverSocket.bind(("localhost", 7979))
10 serverSocket.listen()
11
12 #fonction accepte qui accepte tous les cnx et retourne socket du client et son adresse
13 (client, adrr) = serverSocket.accept()
14 while (True):
15     login = client.recv(1024)#récupérer login
16     print(login)#affichage login
17     concat = b"finger" + b' ' + login #concaténation du "finger" avec la chaîne de caractères reçus(lo
18
19     if login != b'':#si login <> du vide
20
21         out = subprocess.getoutput(concat)#le module commands n'exist plus en python3
22         #il est donc remplacé par celui du subprocess
23
24         client.send(str.encode(out))#envoyer au client out
25     else:
26         serverSocket.close()
27         print("login vide")
28         break
29
30 except socket.error as error:
31     if error.errno == errno.ECONNREFUSED:
32         print(os.strerror(error.errno))
33     else:
34         raise
```

9. Écrivez une fonction qui écrit le pid du processus en cours dans un fichier. Ce fichier pourra être, par exemple, /tmp/finger.pid. Appelez cette fonction à l'initialisation de votre serveur.

10. Modifiez votre serveur pour journaliser les requêtes reçues dans le fichier de logs.

```
while (True):
    login = client.recv(1024)#récupérer login
    print(login)#affichage login
    concat = b"finger" + b' ' + login #concaténation du "finger" avec la chaîne de caractères reçus(login)

    if login != b'':#si login <> du vide

        out = subprocess.getoutput(concat)#le module commands n'existe plus en python3
        #il est donc remplacé par celui du subprocess

        client.send(str.encode(out))#envoyer au client out

    fichier1 = open("/tmp/finger.log", "a")#ouvrir le fichier "/tmp/finger.log"
    fichier1.write(str(datetime.datetime.now()))
    fichier1.write(" : ")
    fichier1.write(login.decode())
    fichier1.write(str(addr[0]))
    fichier1.write(str(addr[1]))
    fichier1.write("\n")
    fichier1.close()
else:
    serverSocket.close()
    print("login vide")
    break
```

11. Écrivez un client qui se connecte sur le serveur fingerd.py, lui envoie une chaîne de caractère, reçoit une réponse et affiche le résultat reçu.

```
finger.py x Client.py x
finger > Client.py > ...
1 import socket
2 import subprocess
3
4 clientSocket= socket.socket()
5 clientSocket.connect(("localhost", 7979))
6
7 login =subprocess.getoutput("whoami")
8
9 clientSocket.send(str.encode(login))
10 rep = clientSocket.recv(1024)
11 print(rep)
12
13
14 verif = "échange terminé"
15 clientSocket.send(str.encode(verif))
```


12. Les arguments de la ligne de commande d'un programme peuvent être récupérés avec la fonction `getopt()` du module `getopt`. Modifiez votre client de façon à permettre à l'utilisateur de passer le nom d'utilisateur à demander et le nom de la machine serveur en paramètres, comme suit :

```
Client.py x
finger > Client.py > ...
1  import socket
2  import sys
3  #python3 Client.py -l effy -f effy
4
5
6  clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7  clientSocket.connect(("localhost", 7979))
8  print("Connexion ...")
9  login = None
10 host = None
11 argv = sys.argv[1:]
12 #récupère les arguments en entrée de l'exécution
13 opts, args = getopt.getopt(argv, "f:l:")#getopt: récupère les arguments sous la forme d'une liste
14 for opt, arg in opts:
15     if opt in ['-f']:
16         login = arg
17     elif opt in ['-l']:
18         host = arg
19 print("login = " ,login)
20 print("host = " , host)
21 if login == '':
22     print("Vérifier login vide !")
23     sys.exit(1)
24
25 size = 0
26 clientSocket.send(str.encode(login))
27 maxSize = int(clientSocket.recv(10000).decode())
28 clientSocket.send(str.encode("ok"))
29
30 while size < maxSize:
31     print(str(clientSocket.recv(1024).decode()))
32     size += 1024
33
```

3 RPC

Exercice 3.1 : Calculatrice RPC

1. Dans un fichier `serveur.py`, créez une classe `Calcul` dans laquelle vous devez implémenter les fonctions du serveur.

```
RPC > Serveur.py > Calcul
1 class Calcul:
2     def add(self,a,b):
3         try:
4             return a+b
5         except TypeError:
6             print("Merci d'utiliser des chiffres")
7             return None
8
9     def mult(self,a,b):
10        try:
11            return a*b
12        except TypeError:
13            print("Merci d'utiliser des chiffres")
14            return None
15
16    def diff(self,a,b):
17        try:
18            return a-b
19        except TypeError:
20            print("Merci d'utiliser des chiffres")
21            return None
22
23    def quotion(self,a,b):
24        try:
25            return a/b
26        except ZeroDivisionError:
27            print("Merci de ne pas diviser par 0")
28            return None
29        except TypeError:
30            print("Merci d'utiliser des chiffres")
31
```

```
31
32    def absolue(self,a):
33        try:
34            return abs(a)
35        except TypeError:
36            print("Merci d'utiliser des chiffres")
37            return None
38
39
```

2. Complétez le programme serveur afin de lancer le serveur RPC lorsque le programme est exécuté. Le serveur doit accepter des connexions depuis n'importe quel client, et écouter sur un port de votre choix.

```
RPC > Exercice 3.1: Calculatrice RPC > Serveur.py > ...
22 def diff(x, y):
23     try:
24         return x-y
25     except TypeError:
26         print("Merci d'utiliser des chiffres")
27         return None
28
29 #Fonction quotient
30 def quotient(x, y):
31     try:
32         return x/y
33     except TypeError:
34         print("Merci d'utiliser des chiffres")
35         return None
36     except ZeroDivisionError:
37         print("Merci de ne pas diviser par 0")
38         return None
39
40 #Fonction Absolu
41 def absolue(x):
42     try:
43         return abs(x)
44     except TypeError:
45         print("Merci d'utiliser des chiffres")
46         return None
47
48
49
50 server = SimpleXMLRPCServer(("localhost", 8085))
51
52 print("Serveur en écoute")
53
54 server.register_function(Calcul.add, 'add')
55 server.register_function(Calcul.mult, 'mult')
56 server.register_function(Calcul.diff, 'diff')
57 server.register_function(Calcul.quotient, 'quotient')
58 server.register_function(Calcul.absolue, 'absolue')
59
60 server.serve_forever()
```

3. Écrivez un client qui se connecte à votre serveur en utilisant le module RPC rfoo et appelle à distance les fonctions que celui-ci implémente.

```
RPC > Exercice 3.1: Calculatrice RPC > Client.py > ...
1 import xmlrpc.client
2
3 try:
4     proxy = xmlrpc.client.ServerProxy("http://localhost:8085/")
5
6     ans=True #variable pour entrer dans la boucle while
7
8     while ans and proxy:
9
10
11
12     # Afficher Menu
13     print("----Menu Calculatrice----")
14     print("1- Addition")
15     print("2- Multiplication")
16     print("3- Différence")
17     print("4- Quotient")
18     print("5- Absolu")
19     print("0- Exit")
20     print(".....")
21
22     Operation=input("Choisir l'opération : ") #Saisir le choix de l'operation
23
24     try :
25         Operation=int(Operation)
26
27         if Operation<5 and Operation>0:
28
29             x=float(input("Entrer valeur x ")) #Saisir valeur1
30             y=float(input("Entrer valeur y ")) #Saisir valeur2
31
32
33             if Operation==1 :
34                 print("Résultat de l'addition est : ")
35                 print(proxy.add(x,y))
36
37             elif Operation==2 :
38                 print("Résultat de la multiplication est : ")
39
```

```
RPC > Exercice 3.1 : Calculatrice RPC > Client.py > ...
37
38     elif Operation==2 :
39         print("Résultat de la multiplication est : ")
40         print( proxy.mult(x,y))
41
42
43     elif Operation==3 :
44         print("Résultat de la différence est : ")
45         choix=input("Si X-Y entrez 1 Si Y-X entrez 2")
46         if choix=="x":
47             print( proxy.diff(x,y))
48         elif choix=="y":
49             print( proxy.diff(y,x))
50         else :
51             print("erreur")
52
53
54     elif Operation==4 :
55         print("Résultat de la quotient est : ")
56         choix=input("Si X\Y entrez 1 Si Y\X entrez 2")
57         if choix=="1":
58             print( proxy.quotient(x,y))
59         elif choix=="2":
60             print( proxy.quotient(y,x))
61         else :
62             print("erreur")
63
64
65
66
67     elif Operation==5 :
68         x=float(input("Entrer valeur 1 "))
69
70         print("Résultat de la valeur absolue du x est : ")
71         print( proxy.absolue(x))
72
73
74     elif Operation==0 :
75         print("Quitter....")
76         ans=False
77
78
79     else :
80         print("Erreur de choisir....")
81
82 except :
83     print("Veuillez insérer un chiffre valide")
84
85 except xmlrpc.client.Fault as err:
86     print("Fault code: %d" % err.faultCode)
87     print("Fault string: %s" % err.faultString)
```

Exercice 3.2 : Annuaire en ligne

1. Écrivez un serveur RPC utilisant une classe Annuaire qui implémente les fonctions suivantes :

```
Seurveur.py X
RPC > Exercice 3.2:Annuaire en ligne > Seurveur.py > Annuaire
1 from xmlrpc.server import SimpleXMLRPCServer
2 i=0
3
4 class Annuaire:
5     #La fonction ajouterEntree
6     def ajouterEntree(Nom, NumeroDeTelephone):
7         global i #declarer i comme une variable globale
8         i += 1 #Avancer le compteur
9         Repertoire=open("Repertoire.txt", 'a') #Ouvrir le fichier Repertoire.txt
10        Repertoire.write("["+Nom+" "+" "+"NumeroDeTelephone+"]"+"\\n") #Ajouter un nom et un numero de telephone
11        Repertoire.close #Fermer le fichier Repertoire.txt
12        return "Element ajoute"
13
14    #La fonction trouverNumero
15    def trouverNumero(nom):
16        global i #declarer i comme une variable globale
17        i += 1 #Avancer le compteur
18        Repertoire=open("Repertoire.txt") #Ouvrir le fichier Repertoire.txt
19        ligne=Repertoire.readlines()
20        for l in ligne:
21            if nom in l:
22                annuaire=[]
23                l=l.replace("\\n","")
24                l=l.replace("[","")
25                l=l.replace("","")
26                l=l.split()
27                annuaire.append(l)
28                for elem in annuaire:
29                    return elem[1]
30
```

```
Seurveur.py X
RPC > Exercice 3.2:Annuaire en ligne > Seurveur.py > Annuaire
31 #La fonction trouverNumero
32 def nbNumeros():
33     global i
34     return i #retourne le nombre d'entrees dans le repertoire
35
36
37 #La fonction getAll
38 def getAll():
39     global i
40     i += 1 #Avancer le compteur
41     Repertoire=open("Repertoire.txt",'r') #Ouvrir le fichier Repertoire.txt
42     text=Repertoire.read()
43     return text
44
45
46 #La fonction supprimerEntree
47 def supprimerEntree(nom):
48     global i
49     i += 1 #Avancer le compteur
50     with open("Repertoire.txt","r+") as f:
51         nv_fichier = f.readlines()
52         f.seek(0)
53         for line in nv_fichier:
54             if nom not in line:
55                 f.write(line)
56         f.truncate()
57         return "supprime avec succés"
58
59 #La fonction supprimerTout()
60 def supprimerTout():
61     global i
```

```
58
59 #La fonction supprimerTout()
60 def supprimerTout():
61     global i
62     i += 1 #Avancer le compteur
63     with open("Repertoire.txt","r+") as f:
64         f.truncate(0)
65
66
67 server = SimpleXMLRPCServer(("localhost", 8089))
68
69 print("Seurveur en écoute")
70
71 server.register_function(Annuaire.ajouterEntree, 'ajouterEntree')
72 server.register_function(Annuaire.trouverNumero, 'trouverNumero')
73 server.register_function(Annuaire.nbNumeros, 'nbNumeros')
74 server.register_function(Annuaire.getAll, 'getAll')
75 server.register_function(Annuaire.supprimerEntree, 'supprimerEntree')
76 server.register_function(Annuaire.supprimerTout, 'supprimerTout')
77
78 server.serve_forever()
```

2. Écrivez un client qui affiche le menu suivant et propose à l'utilisateur de saisir une action à effectuer, et appelle par RPC la fonction du serveur correspondante pour manipuler le répertoire contenu dans le serveur.

```
Client.py x
RPC > Exercice 3.2:Annuaire en ligne > Client.py > ...
1  import xmlrpc.client
2
3  proxy = xmlrpc.client.ServerProxy("http://localhost:8089/")
4
5  ans=True
6  try:
7      while ans :
8
9          #-----Menu-----#
10         print("Choix de l'action à effectuer :")
11         print("1: Ajouter une entree dans le repertoire")
12         print("2: Afficher le numero de telephone d'une personne")
13         print("3: Afficher le nombre de numeros enregistres dans le repertoire")
14         print("4: Afficher le contenu de tout le repertoire")
15         print("5: Supprimer du repertoire une personne et son numero")
16         print("6: Effacer tout le contenu du repertoire")
17         print("0: Quitter le programme")
18
19         #-----Choix-----#
20         Operation=input("Choisir l'opération : ") #Saisir votre choix
21         Operation=int(Operation) #Cast choix
22
23         #-----Choix1 : Ajouter un Entree -----#
24         if Operation==1 :
25             Nom = input("Entrez le nom de la personne que vous enregistrez dans l'annuaire ") #Saisir votre Nom
26             Telephone = input("Entrez le numéro de téléphone de la personne ") #Saisir votre téléphone
27             print(proxy.ajouterEntree(Nom, Telephone)) #Appel à la fonction ajouterEntree
28
29         #-----Choix2 : Trouver le Numéro de téléphone -----#
30         elif Operation==2 :
31             nom = input("Entrez le nom de la personne que vous voulez rechercher ") #Saisir le nom
32             print(proxy.trouverNumero(nom)) #Afficher le numéro de téléphone trouvé
```

```
Client.py x
RPC > Exercice 3.2:Annuaire en ligne > Client.py > ...
30     elif Operation==2 :
31         nom = input("Entrez le nom de la personne que vous voulez rechercher ") #Saisir le nom
32         print(proxy.trouverNumero(nom)) #Afficher le numéro de téléphone trouvé
33
34     #-----Choix3 : Le nombre d'entrées dans le répertoire -----#
35     elif Operation==3 :
36         print(proxy.nbNumeros()) #Afficher le nombre d'entrées dans le répertoire
37
38     #-----Choix4 : Afficher Le contenu du répertoire -----#
39     elif Operation==4 :
40         print(proxy.getAll()) #Appel à la fonction getAll() pur afficher le contenu du répertoire
41
42     #-----Choix5 : Supprimer un element -----#
43     elif Operation==5 :
44         nom = input("Entrez le nom de la personne que vous voulez supprimer ") #Saisir le nom
45         print(proxy.supprimerEntree(nom)) #Appel à la fonction supprimerEntree pur supprimer un element du répertoire
46
47     #-----Choix6 : Supprimer toutes les entrées -----#
48     elif Operation==6 :
49         proxy.supprimerTout() #Appel à la fonction supprimerTout pur supprimer le contenu du répertoire
50
51     #-----Choix0 : Quitter le terminal -----#
52     elif Operation==0 :
53         print("Quitter....")
54         ans=False #Quitter la boucle while
55
56     else :
57         print("Erreur de choisir....") #Autre Choix
58
59 except xmlrpc.client.Fault as err:
60     print("A fault occurred")
61
```

4 Transfert de fichiers avec FTP

1 Écriture d'un client FTP

Écrivez un client FTP qui se connecte à un serveur 3, s'authentifie de manière anonyme, affiche la liste des fichiers, se déplace dans un répertoire et télécharge un fichier. Le DTP et le PI peuvent être implémentés dans le même script Python.

```
Client.py x
FTP > Client.py > ...
1 from ftplib import FTP
2 import socket
3
4
5 BUFFER_SIZE = 1024
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8
9 def TCP_S(ip,port):
10     try:
11         s.connect((ip, port))
12         print ("Connexion avec succès")
13         choix=input("\nEnter c_s pour un transfert du client vers le serveur\n")
14         | | | "Enter s_c pour un transfert du serveur vers le client\n")
15
16         if choix.upper() == "C_S":
17             s.send(bytes("Client to serveur",'UTF-8'))
18             Client2Serveur()
19         elif choix.upper() == "S_C":
20             s.send(bytes("serveur to client",'UTF-8'))
21             Serveur2Client()
22         else:
23             print ("Choix invalide")
24
25     except :
26         print("Echec de connexion TCP")
27
28 #affiche liste de fichier
29 def FTP_S(repEnLigne):#exp: ftp1.at.proftpd.org
30     with FTP(repEnLigne) as ftp:
31         ftp.login()#se connecter en tq anonymous
32
```

```

Client.py X
FTP > Client.py > ...
32
33     choix=input("\nEnter down pour télécharger un fichier\n")
34         "Entrez liste pour afficher la liste des fichiers\n")
35     #Télécharger fichier
36     if choix.upper() == "DOWN":
37         nomfichier = 'README.MIRRORS' #ce fichier exist déjà dans la répertoire en ligne
38         fichierLocal = open(nomfichier, 'wb')#wb permet d'ouvrir le fichier en mode write and binary
39         #car on va copier le fichier avec retrbinary
40         ftp.retrbinary('RETR ' + nomfichier, fichierLocal.write, 1024)# copie sur la machine locale
41         fichierLocal.close()
42         ftp.quit()
43
44     #Afficher la liste des fichiers
45     elif choix.upper() == "LISTE":
46         for fichier in ftp.dir():#dir(): fonction affichant tous les elts dans une répertoire en ligne
47             print(fichier)
48     else:
49         print ("Choix invalide")
50
51
52 def Client2Serveur():
53     filetosend = open("test2.txt","r+")#Le mode r+ permet d'ouvrir un fichier
54     #à la fois en lecture et en écriture
55     try:
56         data = filetosend.read(1024)#lire ligne par ligne
57     except:
58         print("problème dans la lecture de data")
59
60     while data:
61         print("Envoi...")
62         data=data.encode()
63         s.send(data)

```

```

Client.py X
FTP > Client.py > ...
65     data = filetosend.read(1024)
66
67     filetosend.close()
68     s.send(b"DONE")
69     print("Envoi terminé avec succès.")
70
71     s.shutdown(2)
72     s.close()
73
74
75
76 def Serveur2Client():
77     data = s.recv(BUFFER SIZE)
78     print("msg du Serveur :", data.decode())
79     s.send(bytes("Connexion avec succès!!", 'UTF-8'))
80     filetodown = open("test1.txt", "w")
81     filetodown.write(data.decode())
82     filetodown.close()
83     print("Reception....")
84     datal = s.recv(1024)
85     if datal == b"DONE":
86         print("Bien reçu.")
87
88     s.close()
89
90
91 while True:
92
93     choix=input("entrez TCP pour connecter sur un serveur TCP\n")
94         "entrez FTP pour acceder à un site ftp\n"
95         "entrez quit pour quitter\n")
96     if(choix.upper()=="TCP"):

```

```

91 while True:
92
93     choix=input("entrez TCP pour connecter sur un serveur TCP\n")
94         "entrez FTP pour acceder à un site ftp\n"
95         "entrez quit pour quitter\n")
96     if(choix.upper()=="TCP"):
97         ip=input("ip = ")
98         port=int(input("port = "))
99         TCP_S(ip,port)
100     elif(choix.upper()=="FTP"):
101         site=input("site = ")
102         FTP_S(site)
103     elif choix.upper() == "QUIT":
104         s.close()
105         print ("client déconnecté")
106         break
107     else:
108         print("choix invalide")

```


2 Écriture d'un serveur TCP

Écrivez un serveur TCP qui comprend un sous-ensemble des commandes disponibles dans le protocole TCP. Votre serveur devra être capable d'authentifier un utilisateur anonyme, de changer de répertoire et de transférer des fichiers entre le client et le serveur (dans les deux sens). Le DTP et le PI peuvent être implémentés dans le même script Python.

```
FTP > Server.py > ...
1  import socket
2
3  BUFFER_SIZE = 1024
4
5  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6  server.bind(("localhost", 6662))# Lier à l'adresse IP et le port
7  server.listen()
8  print("En attente d'une connexion auprès du client..")
9  while True:
10     conn,clientAddress = server.accept()
11     print("Client connecté:" , clientAddress)
12
13
14     choix=conn.recv(1024)
15     if choix == b"Client to serveur":
16         data = conn.recv(BUFFER_SIZE)
17         print("msg du Client :", data.decode())
18         conn.send(bytes("Connexion avec succès!", 'UTF-8'))
19
20         filetodown = open("test1.txt", "w")
21         filetodown.write(data.decode())
22         filetodown.close()
23
24         print("Reception...")
25         datal = conn.recv(1024)
26         if datal == b"DONE":
27             print("Bien reçu.")
28             break
29
30         conn.close()
31
32     #####
```

```
34
35 elif choix == b"serveur to client":
36     filetosend = open("test.txt", "r+")
37     try:
38         data = filetosend.read(1024)
39     except:
40         print("problème dans la lecture de data")
41
42     while data:
43         print("Envoi...")
44         data=data.encode()
45         conn.send(data)
46         data = filetosend.read(1024)
47
48     filetosend.close()
49     conn.send(b"DONE")
50     print("Envoi terminé avec succès.")
51
52     conn.shutdown(2)
53     conn.close()
54
55
56
57 conn.close()
58 server.close()
```

5 Interaction avec un service web : exemple de l'API Twitter

Question 1 : Qu'est-ce que le screen_name ? Comment Twitter identifie-t-il les comptes utilisateurs ?

Le screen_name désigne le nom d'utilisateur Twitter
Dans notre cas screen_name ="ReseauxTelecom "
Twitter identifie les comptes utilisateurs par leurs id et leurs screen_name.

Question 2 : Que contient le champ status ?

le champ status contient plusieurs informations sur le compte utilisateur tels que:
la date de création, l' id...

Question 3 : Combien d'amis (friends) a notre compte de test ? A quoi correspond un "ami" dans Twitter ?

Notre compte de test a 15 amis.
Un "ami" correspond au fait qu'une personne A soit abonnée à une autre personne B tel que B soit aussi abonné à A.

Question 4 : Combien de suiveurs (followers) a-t-il ?

Notre compte de test a 6 followers.

1.1 Installation des outils nécessaires

Question 5 : Installer le module twitter sur votre machine. Décrivez sur votre compte rendu les étapes suivies.

```
effy@effy:~/Desktop$ pip install twitter
Collecting twitter
  Downloading twitter-1.19.3-py2.py3-none-any.whl (50 kB)
    | 50 kB 131 kB/s
Installing collected packages: twitter
Successfully installed twitter-1.19.3
```

Question 6 : Écrire le code pour obtenir en Python les informations sur l'utilisateur ReseauxTelecom Indication : utiliser la méthode `users.show(screen_name=xxx)`

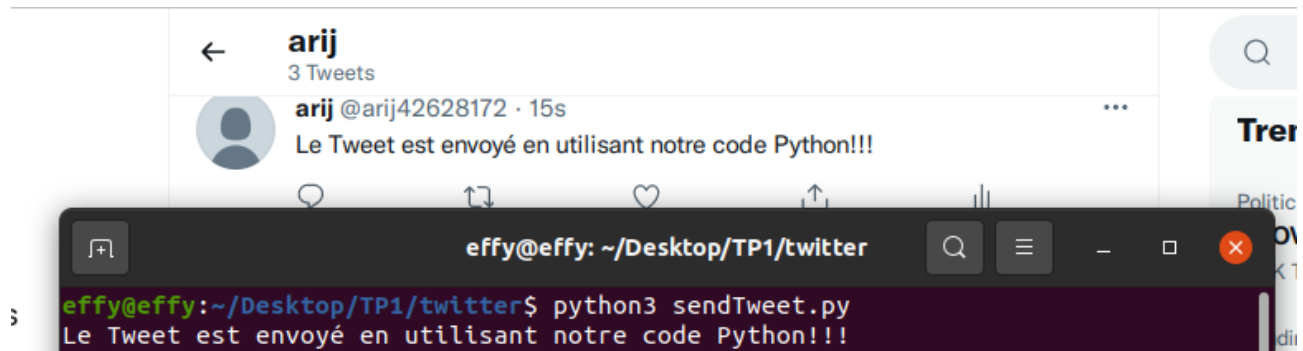
```
ReseauTelecom.py X
twitter > ReseauTelecom.py > ...
1 import tweepy
2
3 consumer_key = 'RPBAX2N91ZW8ec1gu0wppLNZE'
4 consumer_secret = '1zB2pvEb3f8wyVVDkQG1MAFFVQBoodJwKtvLruidtdBugHLGgS'
5 access_token = '1462394680121995265-nxymbakV0u6Mcpe99NqlcPJuDqgCI5'
6 access_token_secret = 'wUi0XQ0Yi7BgU2wFqfJsTWdHzDnqmxUn3wkH0XfxHrLGQ'
7
8 # Authenticate to Twitter
9 def OAuth():
10     try:
11         auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
12         auth.set_access_token(access_token, access_token_secret)
13         return auth
14     except Exception as e :
15         return None
16
17 auth = OAuth()
18
19 # Create API object
20 api = tweepy.API(auth)
21
22 try:
23     api.verify_credentials()
24     print("Authentication OK")
25     user = api.get_user(screen_name="ReseauxTelecom")
26     print("User details:")
27     x= str(user).split(",")
28     for a in x:
29         print(a+'\n')
30
31 except:
32     print("Error during authentication")
```

1.2 Envoi d'un tweet

Question 7 : Écrire le code pour envoyer un tweet sur votre compte.

```
EXPLORER ... sendTweet.py X
TP1
> .vscode
> finger
> FTP
> RPC
> TCP
> twitter
  sendTweet.py
> UDP
  test2.txt

twitter > sendTweet.py > ...
1 import tweepy
2
3 consumer_key = 'RPBAX2N91ZW8ec1gu0wppLNZE'
4 consumer_secret = '1zB2pvEb3f8wyVVDkQG1MAFFVQBoodJwKtvLruidtdBugHLGgS'
5 access_token = '1462394680121995265-nxymbakV0u6Mcpe99NqlcPJuDqgCI5'
6 access_token_secret = 'wUi0XQ0Yi7BgU2wFqfJsTWdHzDnqmxUn3wkH0XfxHrLGQ'
7 def OAuth():
8     try:
9
10         auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
11         auth.set_access_token(access_token, access_token_secret)
12         return auth
13     except Exception as e :
14         return None
15
16 auth = OAuth()
17
18 api = tweepy.API(auth)
19 api.update_status("Le Tweet est envoyé en utilisant notre code Python!!!")
20 print("Le Tweet est envoyé en utilisant notre code Python!!!")
```



1.3 Construction du graphe social autour d'un utilisateur

Question 8 : Écrire un programme en python qui construise et enregistre dans un fichier le “graphe social” d’un utilisateur donné (qui sera identifié par son screen_name), c’est à dire la liste des relations entre cet utilisateur et ses amis, et de ses amis entre eux

```
twitter > graphe_social.py > ...
1 from networkx.classes.function import edges
2 import tweepy
3 import networkx as nx
4 import pandas as pd
5 import matplotlib.pyplot as plt
6
7
8 consumer_key = 'RPBAX2N91Zw8ec1gu0wppLNZE'
9 consumer_secret = '1zB2pvEb3f8wyVVDkQG1MAFFVQ8oodJwKtvLruidtdBugHLGgS'
10 access_token = '1462394680121995265-nxymbakV8u6Mcpe99NqLcPJuDqqCI5'
11 access_token_secret = 'wU10XQ0Y17BgU2wFqfJ5TwDHzDnqmxUn3wkH0XfxHrLGQ'
12
13
14 auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
15 auth.set_access_token(access_token, access_token_secret)
16 api = tweepy.API(auth, wait_on_rate_limit=True)
17
18
19 me = api.get_user(screen_name = 'arij42628172')
20 me.id
21
22 user_list = [1462394680121995265]
23 follower_list = []
24 for user in user_list:
25     followers = []
26     try:
27         for page in tweepy.Cursor(api.get_follower_ids, screen_name = 'arij42628172').pages():
28             followers.extend(page)
29             print(len(followers))
30     except tweepy.TwitterServerError:
31         print("error")
32         continue
```

```
twitter > graphe_social.py > ...
33 follower_list.append(followers)
34
35 fichier = open("followers.txt", "w")
36 for x in followers:
37     print(x)
38     fichier.write(str(me.id)+" "+str(x)+"\n")
39 fichier.close()
40
41 df = pd.DataFrame(columns=['source', 'target'])
42 df['target'] = follower_list[0]
43 df['source'] = 1462394680121995265
44
45 G = nx.from_pandas_edgelist(df, 'source', 'target')
46 pos = nx.spring_layout(G)
47
48
49 f, ax = plt.subplots(figsize=(10, 10))
50 plt.style.use('ggplot')
51 nodes = nx.draw_networkx_nodes(G, pos, alpha=0.8)
52
53
54
55 nx.draw_networkx_labels(G, pos, font_size=8)
56 nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.2)
57
58 plt.show()
59 print(G)
```