# TP 1 Introduction à la programmation réseau

*TP Réseaux Avancés M1 SIC - IA*

Ilyas Bambrik

2024

# Table des matières

# Objectifs

- Analyser l'état des connexions de processus;
- Travail à rendre : pour chaque question/étape prenez une capture d'écran *complète (les captures d'écran partielles ne seront comptabilisées)*.

# Introduction

Afin de réaliser le TP, vous devez télécharger et installer Python 3 ( *https://www.python.org/downloads/release/python-3120/* ) ou avec Jupyter Notebook et Anaconda)

Les codes présentés dans ce support sont téléchargeables à partir du lien suivant *https://drive.google.com/drive/u/0/folders/1XDPqzfoa8K7gOWcoOtbAqpXK7MYX4z9H*

# Exercice : netstat

I

Lancez votre invité de commande (*CMD*) et exécutez la commende suivante pour afficher les connexions établies entre les applications dans votre machines et d'autres processus dans des machines distantes:

```
netstat -n
```

*Remarque :* Une connexion peut être établie entre deux applications qui fonctionnent dans la même machine.

```
Proto  Adresse locale         Adresse distante       État
TCP    127.0.0.1:61558        127.0.0.1:61559        ESTABLISHED
TCP    127.0.0.1:61559        127.0.0.1:61558        ESTABLISHED
TCP    127.0.0.1:61562        127.0.0.1:62556        ESTABLISHED
TCP    127.0.0.1:61605        127.0.0.1:61562        TIME_WAIT
TCP    127.0.0.1:61612        127.0.0.1:61613        ESTABLISHED
TCP    127.0.0.1:61613        127.0.0.1:61612        ESTABLISHED
TCP    127.0.0.1:61614        127.0.0.1:61615        ESTABLISHED
TCP    127.0.0.1:61615        127.0.0.1:61614        ESTABLISHED
TCP    127.0.0.1:62556        127.0.0.1:61562        ESTABLISHED
TCP    127.0.0.1:62562        127.0.0.1:5357         TIME_WAIT
TCP    192.168.1.2:62012      23.35.212.54:443       CLOSE_WAIT
TCP    192.168.1.2:62557      40.69.216.129:443      ESTABLISHED
TCP    192.168.1.2:62558      93.186.135.58:80       TIME_WAIT
```

**Question 1**

Afin d'analyser facilement le résultat de la commande, redirigez le résultat de netstat vers un fichier de sortie (par exemple : *netstat -n>FichierSortie.txt*)

- Exécutez le programme dans *Listing 1* (Serveur.py) avant de lancer le programme de *Listing 2* (Client .py
- Exécutez la commende *netstat -n* à nouveau et vérifiez s'il y a un nouveau port ouvert correspondant au port de votre application Serveur (9500).

```python
import socket

SocketServeur = socket.socket()
port = 9500
# creation d'un socket en ecout sur l'interface 127.0.0.1, port=9500
SocketServeur.bind(("127.0.0.1", port))
SocketServeur.listen(1) # nombre de maximale de connexion qui peuvent etre mis en file d'attente

print( "Lancement serveur") # instruction d'affichage
while True: # boucle infinie
   # accept bloque le programme en attendant une connexion d'un client (la reception du SYN)
   ConnexionAUnClient, addrclient = SocketServeur.accept()
   # une fois que la connexion est recue, accept renvoie l'adresse du client et un objet socket
   # permetant l'envoie et la reception sur cette connexion
   print("Connexion de la machine = ", addrclient)
   MessageRec=""
   while(MessageRec.strip().lower()!="fin"):# tanque le message recu est different de "fin"
      MessageRec=ConnexionAUnClient.recv(1024) # recv permet de recevoir une sequence d'octets
      print( MessageRec.decode())
      # .decode() transforme une objet bytes (suite d'octets) en chaine de caracteres en

   print( "Deconnexion de :",addrclient)
   #ConnexionAUnClient.close() # coloture la connexion (envoie FIN ACK au client)


```

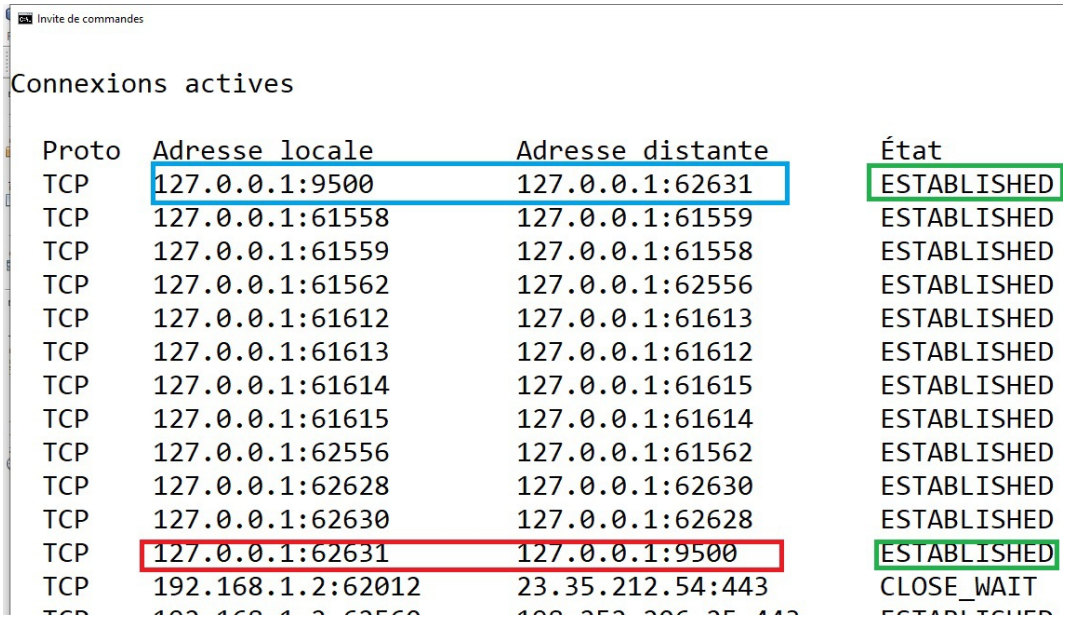
*Listing 1* Programme serveur python

```python
import socket

ConnexionAUnServeur = socket.socket() # Creation de socket
host = "127.0.0.1" # adresse de la machine distante
port = 9500 # numero de port
print( "Console Client")
ConnexionAUnServeur.connect((host, port)) # connexion avec le serveur ( envoie flag SYN)
Message_A_Transmettre=""
while Message_A_Transmettrelower()!="fin":
    Message_A_Transmettre=input() # lecture depuis le clavier
    ConnexionAUnServeur.send(Message_A_Transmettre.encode()) # transmettre le message vers le serveur
    # .encode() transforme une chaine de caracteres en bytes (suite d'octets)
ConnexionAUnServeur.close() # cloture de la connexion avec le serveur (envoie flag FIN)
print( "Fin du programme")

```

*Listing 2* Programme client



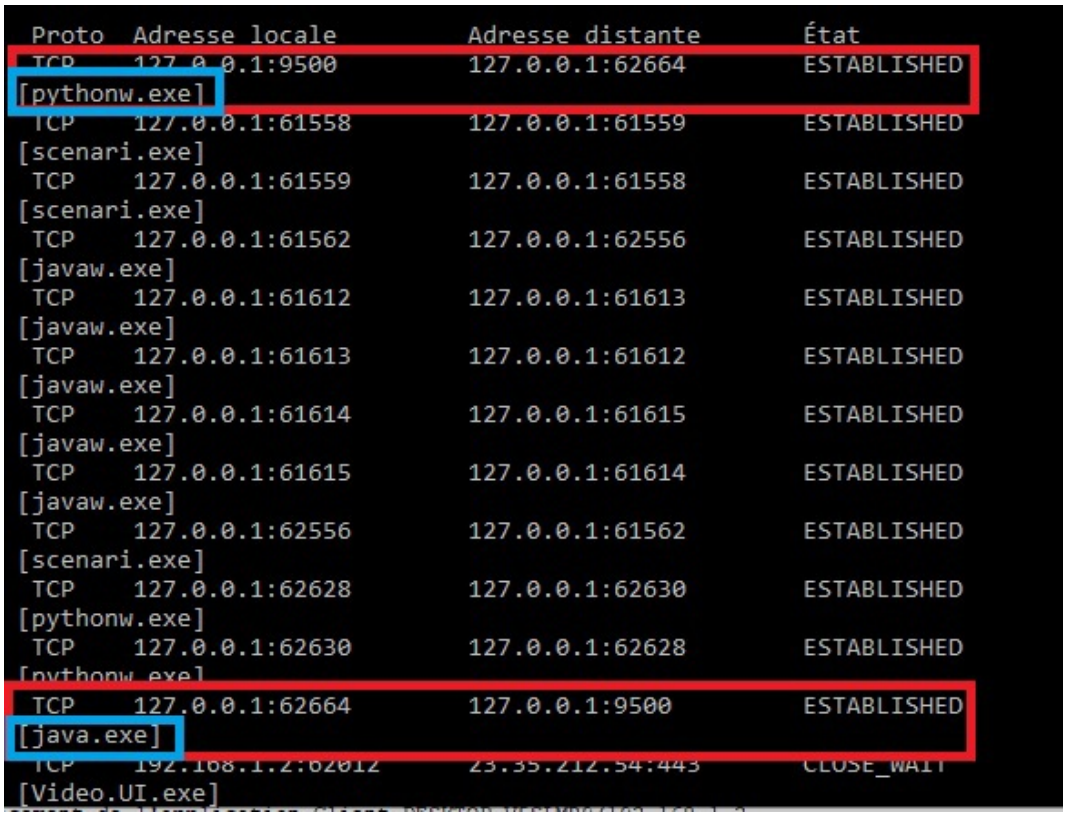

- *Quel est l'état de la connexion avec le serveur (voir la dernière colonne de l'affichage netstat)?*
- *Transmettez un message du client pour le recevoir sur le serveur (écrivez un message dans la console Client et appuyez sur Enter).*

Afin de terminer un processus occupant un numéro de port cible (comme notre serveur par exemple), exécutez les deux commandes suivantes :

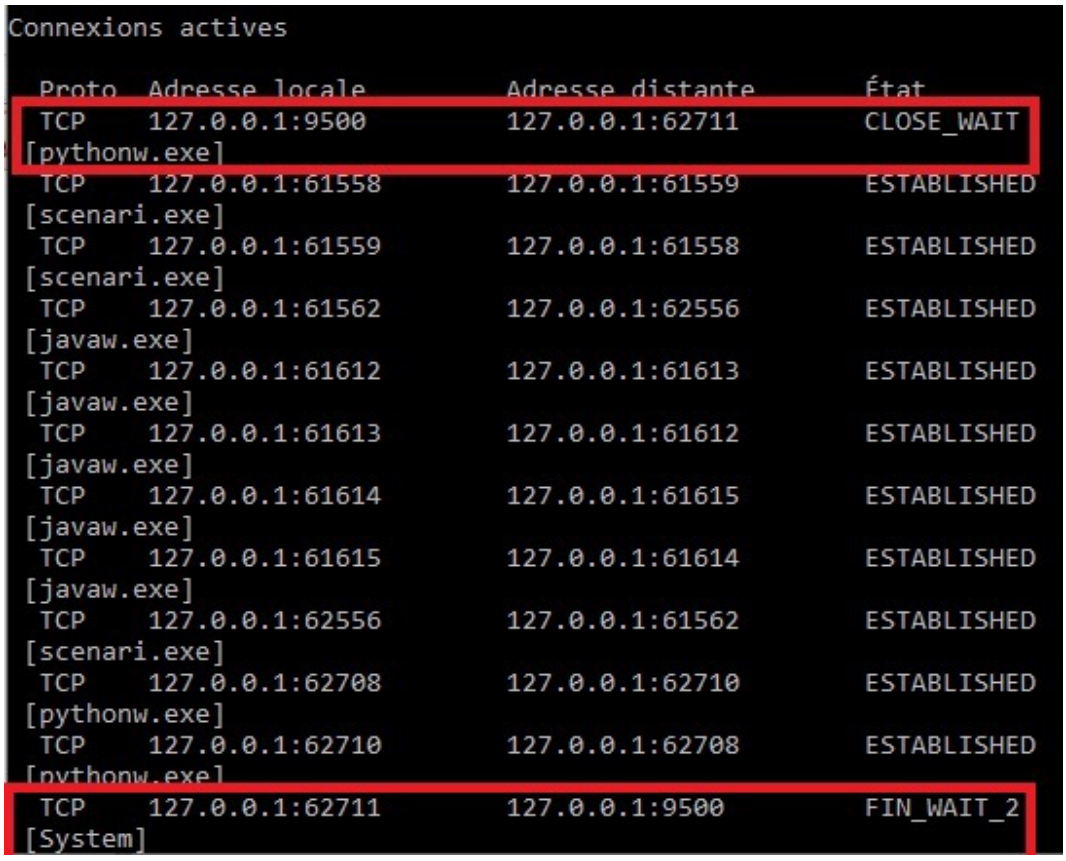
- Trouvez le PID du processus occupant le numéro de port ciblé (dans ce cas 9500): *netstat -aon|findstr : 9500 ;*
- Terminer le processus avec le PID trouvé. Par exemple, pour terminer le processus 1700: *taskkill /PID 1700 /F ;*

**Question 2**

- Lancez votre CMD en *mode administrateur* et exécutez la commende *netstat -n -b* pour voir les nom des processus correspondants à chaque connexion.




- Transmettez un message *"Fin"* de votre client active pour clôturer la connexion et ensuite lancez *netstat -nb* pour voir l'état de connexion entre client et serveur.



- Quel est l'état de la connexion avec le serveur *après la clôture de la connexion*?

**Question 3**

- Lancez deux instances du client avec une seule instance du serveur. Transmettez ensuite des messages a partir des deux clients.
- Que remarquez-vous (est ce que les messages des deux clients sont reçus)?

# Exercice : Multi-threading

II

Le programme serveur précédant peut servir un seul client à la fois. Un serveur web par exemple doit servir plusieurs interlocuteurs simultanément. Ceci est réalisable grâce à la programmation *multi-threading*. Le programme python dans Listing 3 est un simple serveur multithread qui lance un processus chaque fois qu'un nouvel client est connecté.

**Question**

- Lancez le programme client (*Listing 2 du 1er exercice*) et puis le programme serveur multi-thread (*Listing 3*) ;
- Lancez un nouvel client sans fermer le 1er client et transmettez des messages depuis le 1er et 2em client (vérifiez que les messages des deux clients sont reçus par le serveur multithread python)
- Exécutez la commande *netstat -n*;
- *Que remarquez-vous ?*

```python
import socket
import _thread

def Traiter_Connexion(connexion_avec_client,adresse_client):
   MessageRec=""

   print( "Connexion de la machine = ", adresse_client)
   try:
      while(MessageRec.strip().lower()!="fin"):
          MessageRec=connexion_avec_client.recv(1024)
          MessageRec=MessageRec
          print( "Client" ,adresse_client," a dit :",MessageRec.decode())
   except:
      print( "Deconnexion")
   print( "Deconnexion de :",adresse_client)
   try:
      connexion_avec_client.close()
   except:
      pass

SocketServeur = socket.socket()
host = socket.gethostname()
port = 9500
SocketServeur.bind(("127.0.0.1", port))

SocketServeur.listen(5)

print( "Lancement serveur")
while True:
   ConnexionAUnClient, addrclient = SocketServeur.accept()
   _thread.start_new_thread(Traiter_Connexion,(ConnexionAUnClient,addrclient))
```

*Listing 3* Programme serveur python multi-thread

```
Proto  Adresse locale         Adresse distante       Etat
TCP    127.0.0.1:9500         127.0.0.1:62778        ESTABLISHED
TCP    127.0.0.1:9500         127.0.0.1:62779        ESTABLISHED
TCP    127.0.0.1:61558        127.0.0.1:61559        ESTABLISHED
TCP    127.0.0.1:61559        127.0.0.1:61558        ESTABLISHED
TCP    127.0.0.1:61562        127.0.0.1:62556        ESTABLISHED
TCP    127.0.0.1:61612        127.0.0.1:61613        ESTABLISHED
TCP    127.0.0.1:61613        127.0.0.1:61612        ESTABLISHED
TCP    127.0.0.1:61614        127.0.0.1:61615        ESTABLISHED
TCP    127.0.0.1:61615        127.0.0.1:61614        ESTABLISHED
TCP    127.0.0.1:62556        127.0.0.1:61562        ESTABLISHED
TCP    127.0.0.1:62768        127.0.0.1:62770        ESTABLISHED
TCP    127.0.0.1:62770        127.0.0.1:62768        ESTABLISHED
TCP    127.0.0.1:62778        127.0.0.1:9500         ESTABLISHED
TCP    127.0.0.1:62779        127.0.0.1:9500         ESTABLISHED
```

# Interopérabilité Client / Serveur


A noter que les programmes client/serveur ne doivent pas nécessairement être écrit dans le même langage. Le concept de la création de Socket pour écouter / connecter à un numéro de port est le même dans la plus part des langages.

Pour illustrer ceci, lancez le code Serveur python avant de lancer le clients Java suivant qui fonctionne d'une manière identique au client python :

```
1 /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6 package Test;
7
8 /**
9  *
10  * @author DVSR
11  */
12 import java.net.*;
13 import java.io.*;
14 import javax.swing.JOptionPane;
15
16 public class Client {
17     public static void main(String [] args){
18         String NomDuServeur="127.0.0.1";
19         int port=9500;
20         Socket ConnexionAUnServeur;
21
22         DataInputStream FluxDEntrer;
23         DataOutputStream FluxSortie;
24         String MessageATransmettre;
25
26     try {
27
28             System.out.println("Lancement de l'application Client "+InetAddress.
   getLocalHost());
29             ConnexionAUnServeur = new Socket(InetAddress.getByName(NomDuServeur),
   port);
30             FluxSortie = new DataOutputStream(ConnexionAUnServeur.getOutputStream
   ());
31             FluxDEntrer = new DataInputStream(ConnexionAUnServeur.getInputStream
   ());
32         do{
33               MessageATransmettre=JOptionPane.showInputDialog("Entrez le message
   à transmettre:");
34               byte[] b= new byte[MessageATransmettre.length()];
```

```
35
36              int i=0;
37              for(char c:MessageATransmettre.toCharArray()){
38                  b[i]=(byte)c;
39                  i++;
40              }
41
42              FluxSortie.write(b);
43
44
45          }while(!MessageATransmettre.equalsIgnoreCase("Fin"));
46
47          System.out.println("Deconnexion");
48          ConnexionAUnServeur.close();
49      }
50      catch (Exception e) {
51          System.out.println("Erreur ="+e);
52      }
53      }
54
55 }
```