SAE 3.02

Sarah Grenot RT211

This documentation is a help to understand my python code I will explain how the server - client connection is working and how to implement new functionalities.

Connection client-server:

Client:

```
48  if __name__ =="__main__":
49
50      while msg != "disconnect" and msg != "kill":
51          msg = ""
52          client_socket = socket.socket()
53          client_socket.connect(("127.0.0.1", PORT))
54          print("connecté au serveur")
55
56          while msg != "disconnect" and msg != "kill" and msg != "reset":
57              msg = str(input("message:"))
58              envoie = client_socket.send(msg.encode())
59              data = client_socket.recv(1024).decode()
60              print(data)
61
62          client_socket.close()
```

Figure 1-client code

- This is the basic connection between a server a client in local connection.(line 52 to 54)
- The client send messages and receive server messages and print the receive data.
- We had a little functionalities which are a response when kill, disconnect or reset appears.
- **Kill** allow us to shut down the server and the client
- **Disconnect** shut down only the client socket

- **Reset** shut down the socket ,open a new one and connect itself to the new socket

```
import socket
import threading
PORT=10065
msg=""
data = ""




class client():
    def __init__(self, hostname=int, port=int):
        self.__port = port
        self.__hostname = hostname
        self.__socket = None

    def isConnected(self):
        return (self.__socket != None)
```

Beginning of the code we can see the import

And the class client where I put all of my definition

The first definition is “isconnected” it allow to know if there is a connection on my socket.

```
def connect(self):#faire une connection threader
    self.tConnected = threading.Thread(target=self.connected)
    self.tConnected.start()

def connected(self):
        self.__socket = socket.socket()
        print("En attente de connexion")
        self.__socket.connect((self.__hostname,self.__port))
        print("Connexion établie")

def send(self, msg):#envoie de message pour thread
    self.verrou = threading.Lock()
    self.verrou.acquire()
    self.tSend = threading.Thread(target=self.sended, args=[msg])
    self.tSend.start()
    self.verrou.release()

def waitSend(self):
    self.tSend.join()

def sended(self,msg):
        if self.isConnected():
            #while msg != "disconnect" and msg != "kill" and msg != "reset":
                self.__socket.send(msg.encode())
                msg = self.__socket.recv(1024).decode()
                #print(f"{msg} message client")
                return msg
        else:
            print("pas de connexion")
```

The definition “connect” permit is for create threaded connection and start them

Def “connected” is for creating a simple connection with only one host at the time

Def “send” is for sending messages in the tread

Def “wait send” permit to wait all the tread before sending the messages.

Def sended is for sending a messages in the connection for only one host at the time.

I did not use the definition for the threaded connection they are not well enough to be used.

Server:

```
while msg!="kill":
    msg = ""
    cmd=""
    server_socket = socket.socket()
    server_socket.bind(("127.0.0.1", PORT))
    server_socket.listen(5)
    print('serveur démaré')

    while msg != "kill" and msg != "reset":
        msg = ""
        conn, address = server_socket.accept()
        print("connexion")

        while msg != "disconnect" and msg != "kill" and msg != "reset":
            msg = conn.recv(1024).decode()
            if msg == "ip":
                msg = socket.gethostbyname(socket.gethostname())
                conn.send(msg.encode())
            elif msg == "name":
                msg = socket.gethostname()
                conn.send(msg.encode())
            elif msg == "cpu":
                msg = str(psutil.cpu_percent())
                conn.send(msg.encode())
```

Figure 2-server 1 part

- We can find the system of **kill disconnect** and **reset** again working the same as in the client
- Inside the third loop we can see that where the data are dealt with the server receive a message from the client and process it with different responses.
- Ip will send back the server Ip address
- Name will send back the name of the server
- CPU will return the percentage of the using CPU at the moment

```
        elif msg == "RAM":
            psutil.virtual_memory()  # you can convert
            psutil.virtual_memory()._asdict()
            msg = str(psutil.virtual_memory().percent)
            conn.send(msg.encode())
        elif msg == "OS":
            msg = str(sys.platform)
            conn.send(msg.encode())

        elif msg == "python":
            msg = str(sys.version)
            conn.send(msg.encode())
        else:
            conn.send(msg.encode())
    conn.close()

server_socket.close()
```

Figure 3-server 2 part

- RAM will return the percentage of occupied RAM on the server
- OS will return the OS the server is using (exemple:win32)
- Python the server will send the version of python he is using at the moment.
- Any other messages except from them will be send back to the client the same as they were send and will not be processed.

GUI:

```python
import sys
from PyQt5.QtWidgets import *
from client import client



class interface(QMainWindow):
    def __init__(self):
        super().__init__()
        widget = QWidget()
        self.setCentralWidget(widget)
        grid = QGridLayout()
        widget.setLayout(grid)
        self.__fichier = QLabel("Lecture de fichier")
        self.__nomfichier = QLineEdit("")
        self.__lire = QPushButton("Lire")

        grid.addWidget(self.__fichier, 0, 0)
        grid.addWidget(self.__nomfichier, 0, 1)
        grid.addWidget(self.__lire, 0, 2)

        self.__lire.clicked.connect(self.__lireFichier)
```

This is the beginning of the graphic interfaces.

We can see the class interface has been created with all the import.

We create the with Q what we want on the interfaces and the position where we want them with the grid.addWidget

Th last line is for reading a definition when a button is clicked.

```python
self.__textcmd = QLineEdit("")
self.__cmd= QLabel("ligne de commande:")
self.__validecmd = QPushButton("valider")
self.__disconnect = QPushButton("disconnect")
self.__kill = QPushButton("validation")
self.__reset = QPushButton("reset")
self.__choose = QComboBox()
self.__choose.addItem("OS")
self.__choose.addItem("CPU")
self.__choose.addItem("RAM")
self.__choose.addItem("IP")
self.__choose.addItem("Name")
self.__commande = QComboBox()
self.__commande.addItem("kill")
self.__commande.addItem("reset")
self.__commande.addItem("disconnect")
self.__valider = QPushButton("valider")
self.__affichage= QTextBrowser()
self.__choose.currentIndexChanged.connect(self.selectionchange)

self.__valider.clicked.connect(self.traitement)
self.__kill.clicked.connect(self.__tkt)

grid.addWidget(self.__cmd,1,0)
grid.addWidget(self.__textcmd,1,1)
grid.addWidget(self.__validecmd,1,2)
grid.addWidget(self.__choose,2,1)
grid.addWidget(self.__valider,2,2)
grid.addWidget(self.__affichage,2,3)
#grid.addWidget(self.__disconnect,3,0)
grid.addWidget(self.__kill,3,1)
```

Here is the same as before we create little things to add on our interfaces and we place them and we adding the definition for the button

```
def __lireFichier(self):
    print(f"Lecture du fichier {self.__nomfichier.text()}")
    self.__clientList = []
    IP = []
    IP.append("127.0.0.1")
    #IP.append(self.__nomfichier.text())
    for ip in IP:
        print(f"Connection à {ip}")
        monclient = client(ip, 10065)
        monclient.connected()
        self.__clientList.append(monclient)
        self.__affichage.setText("connecté au serveur")
```

This definition is for reading a file(not properly working)

We write in the blank space the name of the file and it is supposed to read it .

For me I put the local address we can also put a classic long distance connection if we uncomment the grey line.

If we change the connected in connect we can create a threaded connection.

```
def traitement(self,client):
    self.__temp=[]
    for client in self.__clientList:
        if self.__choose.currentText()=="OS":
            msg=client.sended("OS")
            self.__affichage.append(f"os: {msg}")
        if self.__choose.currentText()=="CPU":
            msg = client.sended("CPU")
            self.__affichage.append(f"cpu :{msg}% d'utilisation")
        if self.__choose.currentText()=="RAM":
            msg = client.sended("RAM")
            self.__affichage.append(f"ram: {msg}%")
        if self.__choose.currentText()=="IP":
            msg = client.sended("IP")
            self.__affichage.append(f"adresse ip: {msg}")
        if self.__choose.currentText()=="Name":
            msg=client.sended("Name")
            self.__affichage.append(f"nom du serveur:{msg}")
```

In this definition we most of the work we treat the data and put them on the interfaces we have a little combo box that we put beforehand. In this part of the code in say that if the word in the box is “OS” it will send to server the message OS and the server will send back something that I will put on a blank box on the interfaces.

It will work with other functionalities such as : os, ram, cpu, name and ip

```
    def __tkt(self):
        for client in self.__clientList:
            if self.__commande.currentText()=="kill":
                client.sended("kill")
                self.__affichage.setText("serveur mort")
            if self.__commande.currentText()=="disconnect":
                client.sended("disconnect")
                self.__affichage.setText("deconnexion")
            if self.__commande.currentText()=="reset":
                client.sended("reset")
                self.__affichage.setText("reset de la connexion")


    def selectionchange(self, i):
        print ("Items in the list are :")

        for count in range(self.__choose.count()):
            print (self.__choose.itemText(count))
        print("Current index", i, "selection changed ", self.__choose.currentText())


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = interface()
    window.show()
    app.exec()
```

This is the last part of the code is for the functionalities kill, disconnect, reset.

It will disconnect the sever from the client

Kill the connection(shut down the interfaces)

Or reset it(close the socket dos not reconnect to it must be done by yourself)