Technical Document

Batjari Elmir

Introduction

The goal of this project is to have several servers with a client that communicates with each other. Each server and client executes a script. The client connects with the ip and the port and can execute commands asynchronously. In addition it contains a graphical interface.

The server part.

This part was done as follows:

```
def serveur():
    message_client = ""
    while message_client.lower() != "kill":

        message_client = ""
        while message_client.lower() != "kill":
            server_socket = socket.socket()
            server_socket.bind((host, port))
            server_socket.listen(2)
            print(f"Socket ouverte sur {host} - {port}")

            message_client = ""
            while message_client.lower() != "kill" and message_client.lower() != "reset":
                print('> En attente du client')
                conn, address_client = server_socket.accept()
                print(f'Client connecté {address_client}')
                try:
                    message_client = ""
                    while message_client.lower() != "kill" and message_client.lower() != "reset" and message_client.lower() != "disconnect":

                        message_client = conn.recv(1024).decode()
                        print(f"Message recu {message_client}")
                        execution = execute(message_client)
                        conn.send(execution.encode())
                except:
                    pass
                try:
                    conn.send('DISCONNECT'.encode())
                except:
                    pass
                conn.close()
                print("Fermeture de la socket client")

            server_socket.close()
            print("Fermeture du serveur")


if __name__ == '__main__':
    serveur()
```

In this code everything is played in syntax each step allows to do a certain action which leads to a stop when the condition for example kil is met.
This allows the listener to receive and send commands.
It also fulfills the kill reset and dis connect conditions.

This part is used to send a give to the client so that it closes which prevents me from having errors and closes the client correctly.

Function part of the commands:

```
def help():

    return ("""COMMANDE: --ip-- -- hostname -- -- ram-- -- os -- -- cpu -- -- kill -- -- reset -- -- disconnect \n- ping - PYHTON -VERSION - DOS: - linux:- PowerShell:

def ip():
    name = socket.gethostname()
    ip = socket.gethostbyname(name)
    return f"L'ip de votre machine : {ip}"

def nom():
    name = socket.gethostname()
    return f"Le nom de la machine est : {name}"


def ram():
    cmd = psutil.virtual_memory()
    ram1 = cmd[0] / 1000000000
    ram2 = cmd[1] / 1000000000
    ram3 = cmd[3] / 1000000000
    return f"Memoire Total :{ram1}\n"\
           f"Memoire utilisée :{ram2}\n"\
           f"Memoire libre :{ram3}"

def DOS(cmd):

    try:
        if sys.platform == "win32":
            re = cmd.split(':')[1]
            res = subprocess.check_output(re, shell=True).decode("cp850")
            return res
        else:
            return "DOS n'est pas correcte essayer linux ou PowerShell"
    except:
        return 'Error command'
```

```
def linux(cmd):
    try:
        if sys.platform == "linux" or sys.platform == "linux2":
            x = cmd.split(":", 1)[1]
            res= subprocess.getoutput(x)
            return res
        else:
            return "Linux n'est pas correcte essayer DOS ou PowerShell"

    except:
        return 'Error command'

def powershell(cmd):
    try:
        if sys.platform == "win32":
            shell = cmd.split(":", 1)[1]
            res = subprocess.getoutput('PowerShell -command "' + shell + '"')
            return res
        else:
            return 'PowerShell commands are not recognized try Linux'
    except:
        return 'Error command'



def OS():
    res = sys.platform
    if res == 'win32':
        res = 'La machine est un Windows'
    elif res == 'linux' or res == 'linux2':
        res = 'La machine est un Linux'
    elif res == 'darwin':
        res = 'La machine est un MAC OS '
    return res


def cpu_():
    res = psutil.cpu_percent()
    return f'Capaciter du CPU: {res} %'
```

All this part I define functions allowing me to call later.

To call this function I create the execute function which allows me to structure all its correctly and make sure that just this command is readable.

```
def execute(cmd):
    if cmd == 'cpu':
        res = cpu()
        print(f'voici le cpu de la machine: {res}')

    elif cmd == 'os':
        res = OS()
        print(f"L OS est un {res}")

    elif cmd == 'ram':
        res = ram()
        print(f"ram {res}")

    elif cmd == 'ip':
        res = ip()
        print(f"{res}")

    elif cmd == 'help':
        res = help()
        print(f"{res}")

    elif cmd == 'hostname':
        res = nom()
        print(f"{res}")

    elif cmd[0:4] == "DOS:":
        res = DOS(cmd)
        return f"{res}"

    elif cmd[0:6].lower() == 'linux:':
        res = linux(cmd)
        return f"{res}"

    elif cmd[0:11].lower() == 'powershell:':
        res = powershell(cmd)
        return f"{res}"
    elif cmd[0:4] == "ping":
        re = cmd.split(' ')[1]
        res = subprocess.getoutput(cmd)
        return f"{res}"
    else:
        res = "Unknown Command"
    return str(res)
```

The customer part:

I created two class there class which allows me to have a connection of a server but unique it is to say it will have just the characteristics of the port and ip just has it which allows me thereafter to develop several servers at the customer.

This class contains:

```
def addUI(self):

    self.text = QTextEdit(self)
    self.text.verticalScrollBar().rangeChanged.connect(lambda: self.text.verticalScrollBar().setValue(self.text.verticalScrollBar().maximum()))
    self.text.setReadOnly(True)
    self.text.setGeometry(10, 10, 570, 430)
    self.text.setStyleSheet('background-color:white;)')

    self.text2 = QLineEdit(self)

    self.text2.setPlaceholderText('Envoyer du contenu')
    self.text2.setGeometry(10, 440, 570, 30)
    self.text2.setStyleSheet('background-color:white;)}')
```

My interface for writing therefore controls the tab that opens.

Sending message:

```
def send_msg(self):
    if len(self.text2.text()) > 0:
        msg = self.text2.text()
        self.text.append('MOI : ' + msg + '\n')
        if not self.connectionClosed:
            try:
                print(msg)
                if msg != "":
                    self.connnection.send(msg.encode())
            except:
                self.text.append('Impossible de communiquer a
        else:
            self.text.append('Déconnecté. \n')
        self.text2.clear()
```

Allows you to send the message and display it on the interface.

Receive orders:

```
def recv_msg(self):
    while not self.connectionClosed:
        try:
            data = self.connnection.recv(1024).decode()
            self.text.append(data + '\n')
            if data == 'DISCONNECT':

                self.close()
        except:
            pass
```

Receives commands from the server and if I receive a disconnect it resets and disconnects and closes the client.

The second class for the connection and csv:

Port and IP connection:

```
def __start(self):
    if len(self.__ip.text()) > 0 and self.__port.text().isdigit():
        HOST = self.__ip.text()
        PORT = int(self.__port.text())
        try:
            client = socket.socket()
            client.connect((HOST, PORT))
            tab = shell(connection=client)
            self.__tabs.addTab(tab, f'{HOST}:{PORT}')
            self.connections.append(tab)

        except:

            print('ERREUR DE CONNECTION')
    else:
        print('ERREUR DE FORMULAIRE')
```

Allows you to connect to the server that is listening.

CSV :

```
def create_table(self):
    self.table = QTableWidget()
    self.table.setRowCount(0)
    self.table.setColumnCount(1)

    try:
        with open('test.csv','r') as file:
            lines = file.readlines()
            for line in lines:
                line =line.replace('\n', '')
                if len(line.split(',')) == 2 and line.split(',')[1].isdigit() and len(line.split(',')[0]) > 0:
                    HOST = line.split(',')[0]
                    PORT = line.split(',')[1]

                    row = self.table.rowCount()
                    self.table.setRowCount(row + 1)
                    self.table.setItem(row, 0, QTableWidgetItem(HOST + ':' + PORT))
    except:
        with open('test.csv','w') as file_:
            file.write('')

    self.table.horizontalHeader().setStretchLastSection(True)
    self.table.verticalHeader().setVisible(False)
    self.table.horizontalHeader().setVisible(False)
    self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Fixed)
    self.table.setFocusPolicy(QtCore.Qt.NoFocus)
    self.table.setEditTriggers(QAbstractItemView.NoEditTriggers)
    self.table.resizeRowsToContents()
```

I open by default a csv file where inside we will put the different ip and server port by hand afterwards it will be displayed in the csv tab.