Programmer documentation : SAE3.02

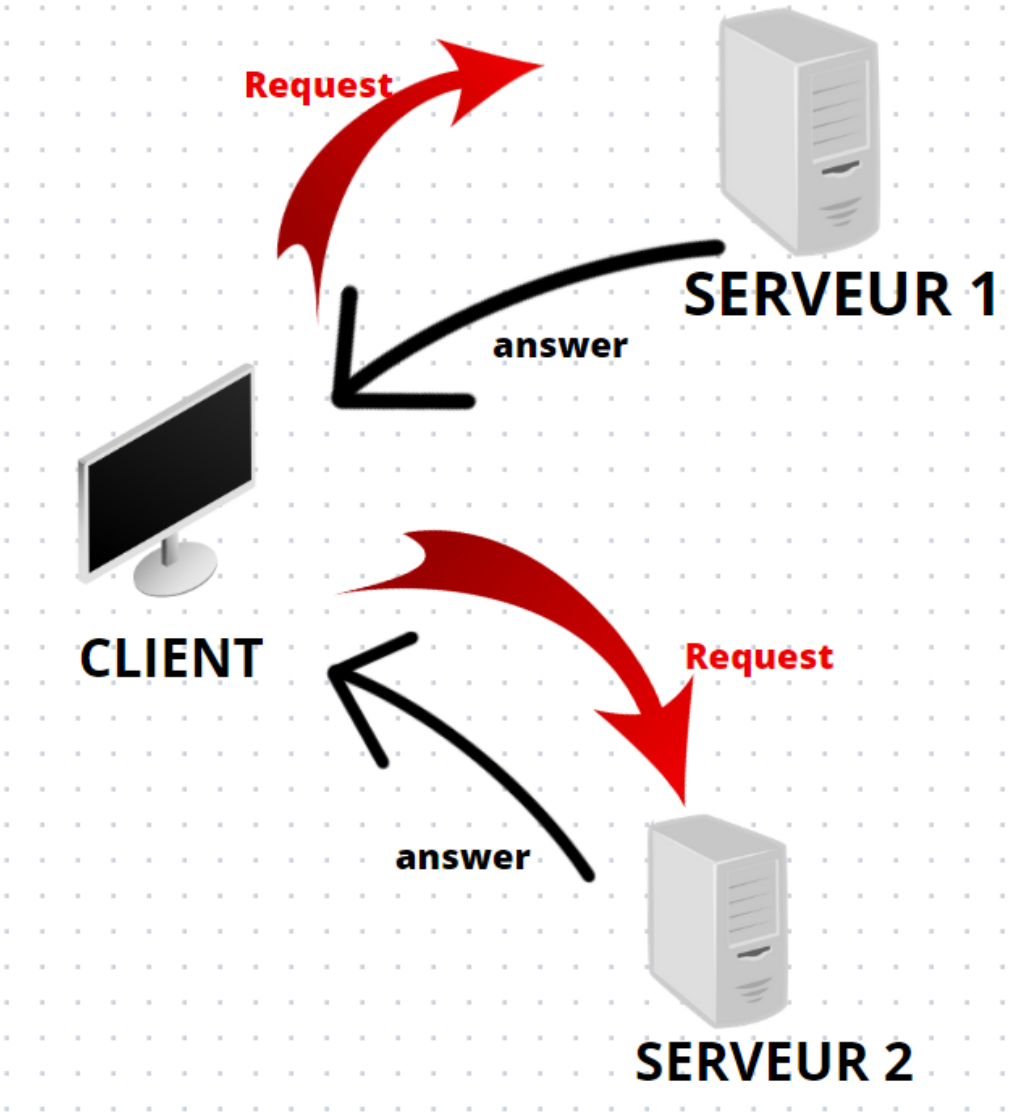
Emir ERASLAN RT23DevCloud

Application’s architecture :

In my application’s architecture there is 1 client and many servers.

The servers are listening waiting while a client connect to them.

Client can connect to one server at once, so if you are connected to serveur 1 for example, you need to disconnect to etablish a new connexion with serveur 2.



1-Architecture

Library :

I use **psutil** to access system details and process utilities.Psutil is a Python cross-platform library. In my application, thanks to psutil, we can get the ram and cpu informations.

I also use **subprocess :**

Subprocess allows me to run external programs. For example, i use it for the ‘ping’ command.

```
elif commande == "ping":
    rep = subprocess.getoutput(data)
    conn.send(rep.encode())
```

Subprocess will help me to execute ‘ping’ command in a shell and he will return the output. Then i can send this output to client.

‘Sys’ and ‘platform’ :

These 2 librairies help me to execute my program in the right operating system.

For example in my program, i have :

```
def Serveur(data):
    if sys.platform == 'win32':
        commande = data.split(' ')[0]
        if commande == "ram":
```

So if the operating system is Windows the program will execute the following lines. The command ‘sys.platform’ will return a string who contains a platform identifier.

For linux, the command will be :

```
if sys.platform == "linux" or sys.platform == "linux2":
    try:
```

Or i use platform.platform for the commande ‘os’ :

```
elif data == "os":
    rep = platform.platform()
    conn.send(rep.encode())
```

Here, platform.platform() will return a string, identifying the platform with as much information as possible.

Code architecture :

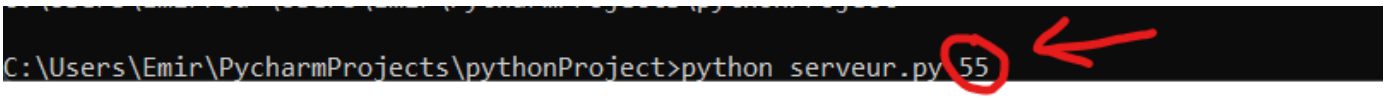
First, we have to code the host and the port configuration to etablish a connection between the client and the server.

```
host = 'localhost'
if len(sys.argv) == 2:
    port = int(sys.argv[1])
else:
    port = 50
```

In my application host is localhost or 127.0.0.1.

The default port for the server is 50, when we run it on Pycharm.

When we want to execute an other server, we just need to add the port that we choose in the command that we execute in cmd for example.



Here, i decide to execute my server on cmd, and i choose the port 55.

For the server, i have 1 function in my code.

The function is ‘Serveur’ :

```
def Serveur(data):
    if sys.platform == 'win32':
        commande = data.split(' ')[0]
        if commande == "ram":
            rep = str(f'RAM utilisé :{psutil.virtual_memory().percent}% \nMémoire vive Total :
{psutil.virtual_memory().total / 1024 / 1024 / 1024:.2f} GB \nMémoire libre restante :
{psutil.virtual_memory().available / 1024 / 1024 / 1024:.2f} GB')
            conn.send(rep.encode())
            print("Information de la ram envoyé au client")
```

First, we verify that we are on the right operating system with ‘sys.platform’.

Then, we can start to code the commands, i start with ‘ram’, if the command that the client enter is ‘ram’ then we answer a string with the psutil tools that we need, like ‘psutil.virtual_memory().percent’ give us the the percent of ram that is used currently.

We send these informations to the client with the line :

‘conn.send(rep.encode)’

I use ‘.encode’ to encode the string because we can’t send data in utf-8, so as we didn’t specify the encoding, utf-8 will be used as default.

In the client, we need to decode the data with ‘.decode’.

« dos : » :

```
elif data[0:4].lower() == 'dos:':
    try:
        cmd = data.split(':', 1)[1]
        output = subprocess.getoutput(cmd)
        conn.send(output.encode())
    except:
        conn.send(f'erreur avec la commande : {data}'.encode())
```

Here, if the first 4 characters are equal to ‘dos:’, the program will collect the characters after the ‘dos:’ and make a subprocess in cmd and he will return the output, then we encode the output and send it to the client.

Socket : kill/reset/disconnect :

```
data = ""
try:
    print("serveur en attente")
    while data != "kill":
        serv_socket = socket.socket()
        serv_socket.bind((host, port))
        serv_socket.listen(5)
        data = ''
        while data != "reset" and data != 'kill':
                conn, address = serv_socket.accept()
                data = ''
                print("client connecté")
                while data != "disconnect" and data != 'kill' and data != 'reset':
                    try:
                        data = conn.recv(1024).decode().lower()
                        Serveur(data)
                    except:
                        pass
                try:
                    conn.send("DISCONNECT".encode())
                    print("deconnexion du client")
                except:
                    pass
                conn.close()
        serv_socket.close()
except ConnectionResetError:
    print("")
```

The server is waiting while listening, with bind we assign an adress/port number local to the socket.

In the second loop ‘while’, the server will accept the connection, and he will receive and decode the data while the message is not disconnect, reset or kill. If the message is kill, the first ‘while’ loop will close the server.

Client :

```python
def __connect(self):
    try:
        if not self.__connection_active:
            host = self.__ip.text()
            port = int(self.__port.text())
            self.client_socket = socket.socket()
            self.client_socket.connect((host, port))
            self.affichage.append(f"Vous êtes dès à présent connecté à {host} port : {port}")
            self.__connection_active = True
            self.ecoute_thread = threading.Thread(target=self.ecoute)
            self.ecoute_thread.start()
        else:
            self.affichage.append("Veuillez d'abord vous déconnecter du server avant de lancer une nouvelle connection!")
    except:
        self.affichage.append(f'Erreur lors de la connection vers le server!!!')
```

With this function, we can connect with the server at the host and the port that the server is running. I launch the thread with self.ecoute.

```python
def ecoute(self):
    while self.__connection_active:
        try:
            data = self.client_socket.recv(1024).decode()
            self.affichage.append(f"{data}")
            if data == 'DISCONNECT':
                self.__connection_active = False
                self.client_socket.close()
                self.affichage.append(f'Déconnecté du serveur')
        except:
            self.affichage.append(f'erreur au niveau de la connection!!!')
```

This function is the program that is listening the server, we receive the data and decode it and then I send it to my graphic interface.

The « .append » help to keep the data in the iterable form.

If data is disconnect then the client socket is close and we are disconnected from the server.

I failed to connect to a client from many servers all active and connected, but I will continue this project as a personal project to finish and learn more about sockets.