

Principiile rețelelor de calculatoare

Socket programming în Java și Python

Conexiunea TCP pune la dispoziție un canal de comunicare dintre un server și un client, prin intermediul căruia informația stocată sub formă de bytes este transmisă între cele două capete ale comunicării. Conexiunea UDP va transmite pachete de informație și nu garantează ordinea în care sunt primite.

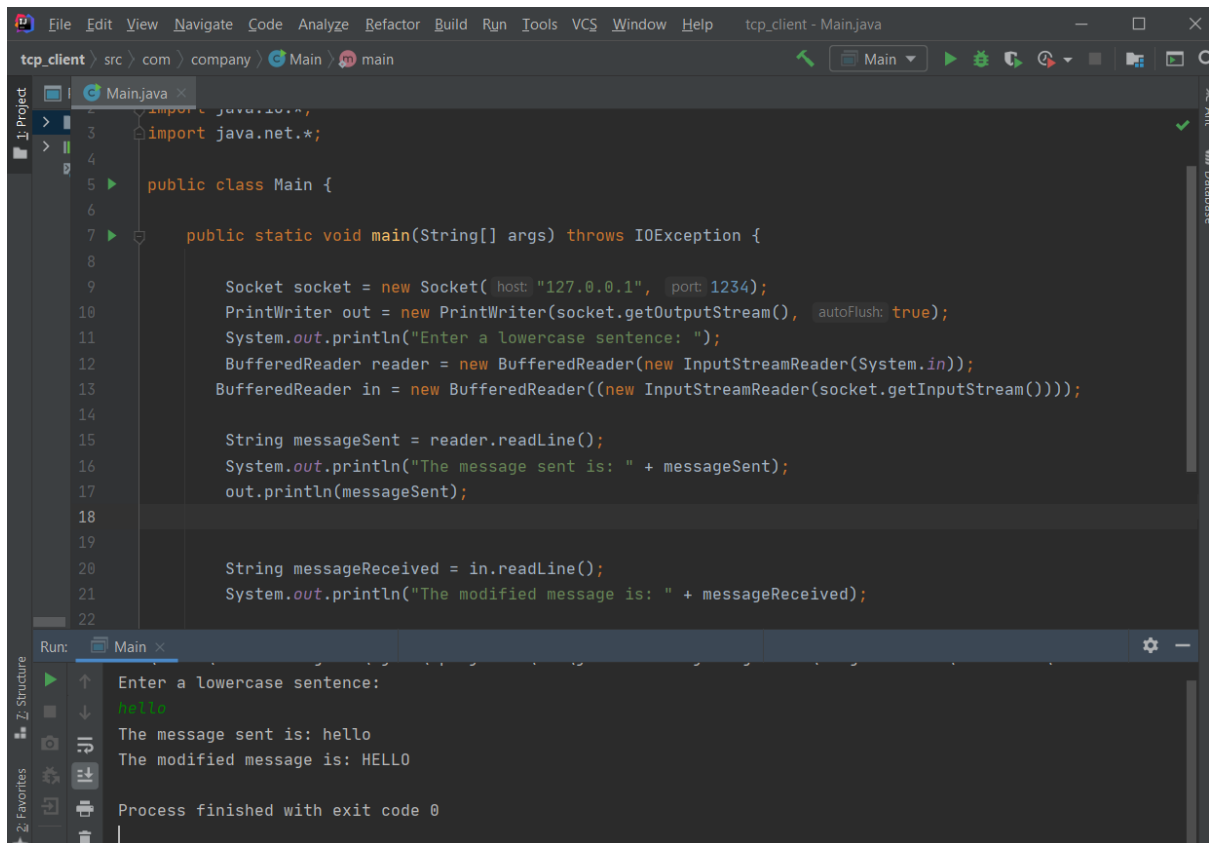
Pentru toate implementările de mai jos, cerința constă în convertirea unui șir de caractere format din litere mici într-un șir care să conțină numai majuscule.

Java socket programming

TCP Server-Client

Pentru crearea unui socket necesar pentru a stabili o conexiune TCP, am avut nevoie de o adresă IP și un număr de port pentru a identifica serverul către care vrem să transmitem datele. Clientul primește și transmite informație prin intermediul unor stream-uri asociate socket-ului, *InputStream* și *OutputStream*, asemănătoare cu cele folosite pentru a citi și afișa mesaje în consolă.

Clientul va transmite serverului la care s-a conectat șirul de caractere citit de la tastatură, urmând să afișeze cuvântul scris doar cu litere mari. Cu ajutorul obiectului *reader* al clasei *BufferedReader*, se citește mesajul de la tastatură. Informația se transmite către server cu ajutorul unui obiect *out*, după ce am apelat funcția de *outstream* pe socket-ul creat. Mesajul modificat, primit ca răspuns din partea serverului, va fi memorat într-o variabilă separată de tip *string* și apoi afișat.



The screenshot displays an IDE window titled 'tcp_client - Main.java'. The code defines a `Main` class with a `main` method that establishes a TCP connection to `127.0.0.1` on port `1234`. It uses `PrintWriter` for sending messages and `BufferedReader` for receiving them. The output console shows the program's execution: it prompts for a lowercase sentence, receives 'hello', prints 'The message sent is: hello', and then prints 'The modified message is: HELLO' after converting the input to uppercase. The process ends with exit code 0.

```
1 import java.io.*;
2 import java.net.*;
3
4
5 public class Main {
6
7     public static void main(String[] args) throws IOException {
8
9         Socket socket = new Socket("127.0.0.1", 1234);
10        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
11        System.out.println("Enter a lowercase sentence: ");
12        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
13        BufferedReader in = new BufferedReader((new InputStreamReader(socket.getInputStream())));
14
15        String messageSent = reader.readLine();
16        System.out.println("The message sent is: " + messageSent);
17        out.println(messageSent);
18
19
20        String messageReceived = in.readLine();
21        System.out.println("The modified message is: " + messageReceived);
22    }
23 }
```

Run: Main x

Enter a lowercase sentence:
hello
The message sent is: hello
The modified message is: HELLO
Process finished with exit code 0

Fig 1.1. TCP Client în Java

După crearea socket-ului în cazul serverului, va trebui să se stabilească conexiunea TCP necesară. Serverul își alocă un socket special pentru a comunica cu clientul - pe care l-am denumit *clientSocket* - și va primi inclusiv adresa clientului prin intermediul său, conexiunea fiind realizată după apelarea metodei *accept()*. Similar cu ce se întâmplă în cazul clientului, serverul va prelua mesajul transmis cu ajutorul acestui socket, va modifica șirul de caractere astfel încât să conțină numai majuscule, apoi va transmite printr-o variabilă separată rezultatul către client. Se folosesc aceleași funcții separate *inputStream* și *outputStream* pentru citirea mesajului de la client și respectiv transmiterea altui mesaj către acesta.

The screenshot shows an IDE window titled 'tcp_server - Main.java'. The code defines a `Main` class with a `main` method. The method prints 'The server is ready', binds a `ServerSocket` to port 1234, accepts a `clientSocket`, and prints its address. It then reads a line from the client, prints it, converts it to uppercase, and prints the modified message. The output window shows the following sequence of events:

```
The server is ready
Address of the client is: Socket[addr=/127.0.0.1,port=51533,localport=1234]
The received message from client: hello
Modified message which is sent to client: HELLO
Process finished with exit code 0
```

Fig 1.2. TCP Server în Java

UDP Server-Client

După aflarea adresei IP a serverului și crearea socket-ului, se va citi de la tastatură un mesaj format din litere mici, pe care îl stocăm într-o variabilă. Întrucât o conexiune UDP constă într-un schimb de date de tip bytes, avem nevoie de doi vectori de tip byte, pentru a afișa mesajul scris de la tastatură, respectiv pentru a afișa varianta sa scrisă cu majuscule. Pentru ca un client să poată trimite mesajul către server, are nevoie de un pachet, pe care l-am numit *sendPacket*, care să conțină informațiile necesare privind conținutul său și adresa de destinație: vectorul și lungimea vectorului care conține mesajul ce trebuie convertit, la care se adaugă numele și portul serverului. Clientul va afișa mesajul modificat după ce primește ca răspuns din partea server-ului un pachet de același tip ca cel pe care l-a trimis. Șirul de caractere convertit trebuie de asemenea să fie memorat într-un vector de tip byte.

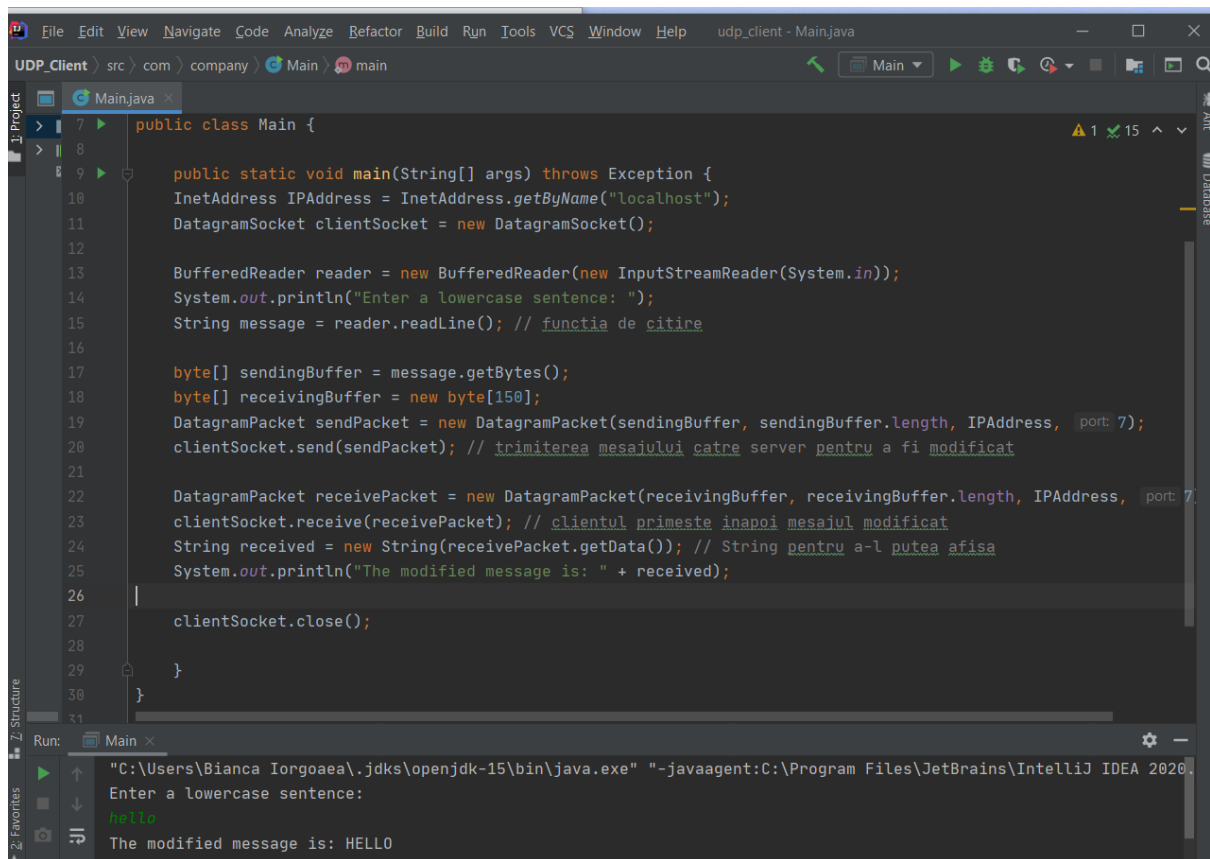


Fig 1.3. UDP Client în Java

Serverul va trebui să aștepte ca un client să se conecteze după crearea socket-ului și comunicarea portului prin care se va realiza conexiunea. Primește pachetul trimis anterior de client și va afișa cuvântul primit după ce a convertit conținutul pachetului într-un șir de caractere. Mesajul modificat va fi inclus în vectorul de tip byte și trimis cu ajutorul unui pachet.

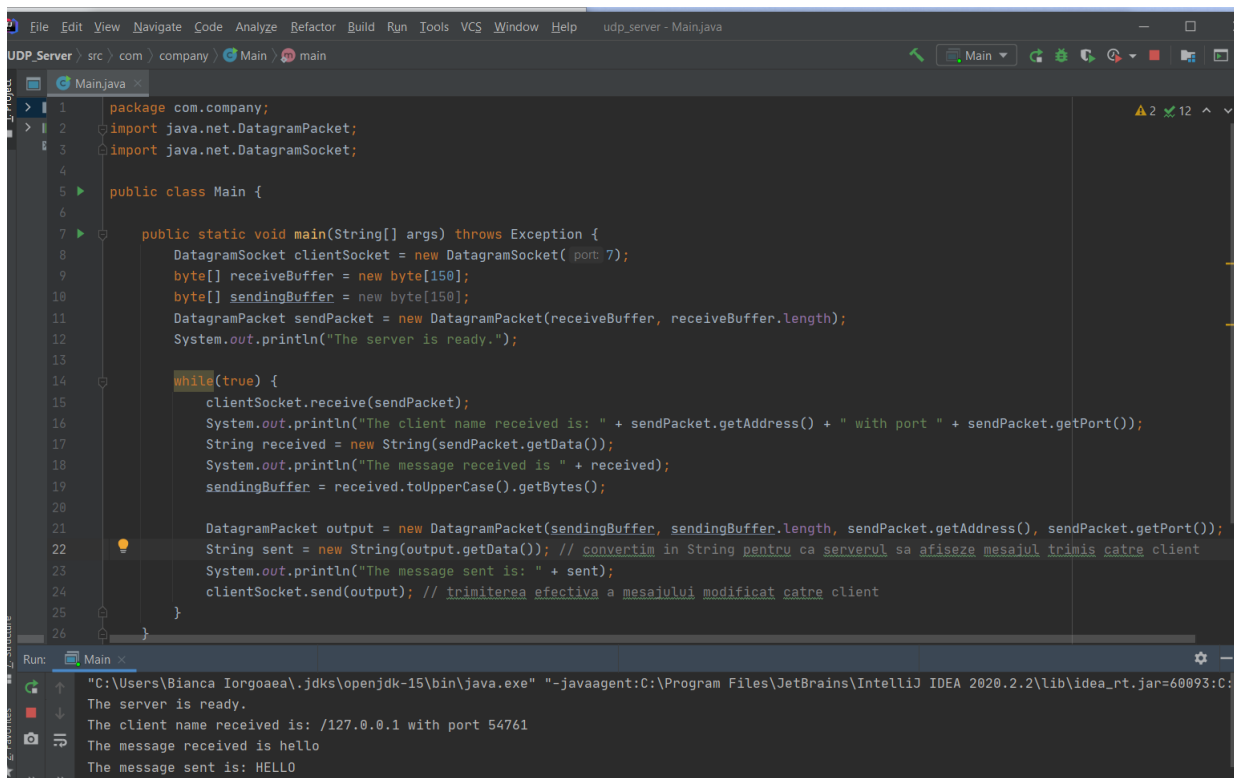


Fig 1.4. UDP Server în Java

Atât în cadrul clientului, cât și în cadrul serverului se folosesc obiecte ale clasei `DatagramPacket`, specifică pentru UDP și funcții ale socket-urilor, `send` și `receive` cu parametri corespunzători.

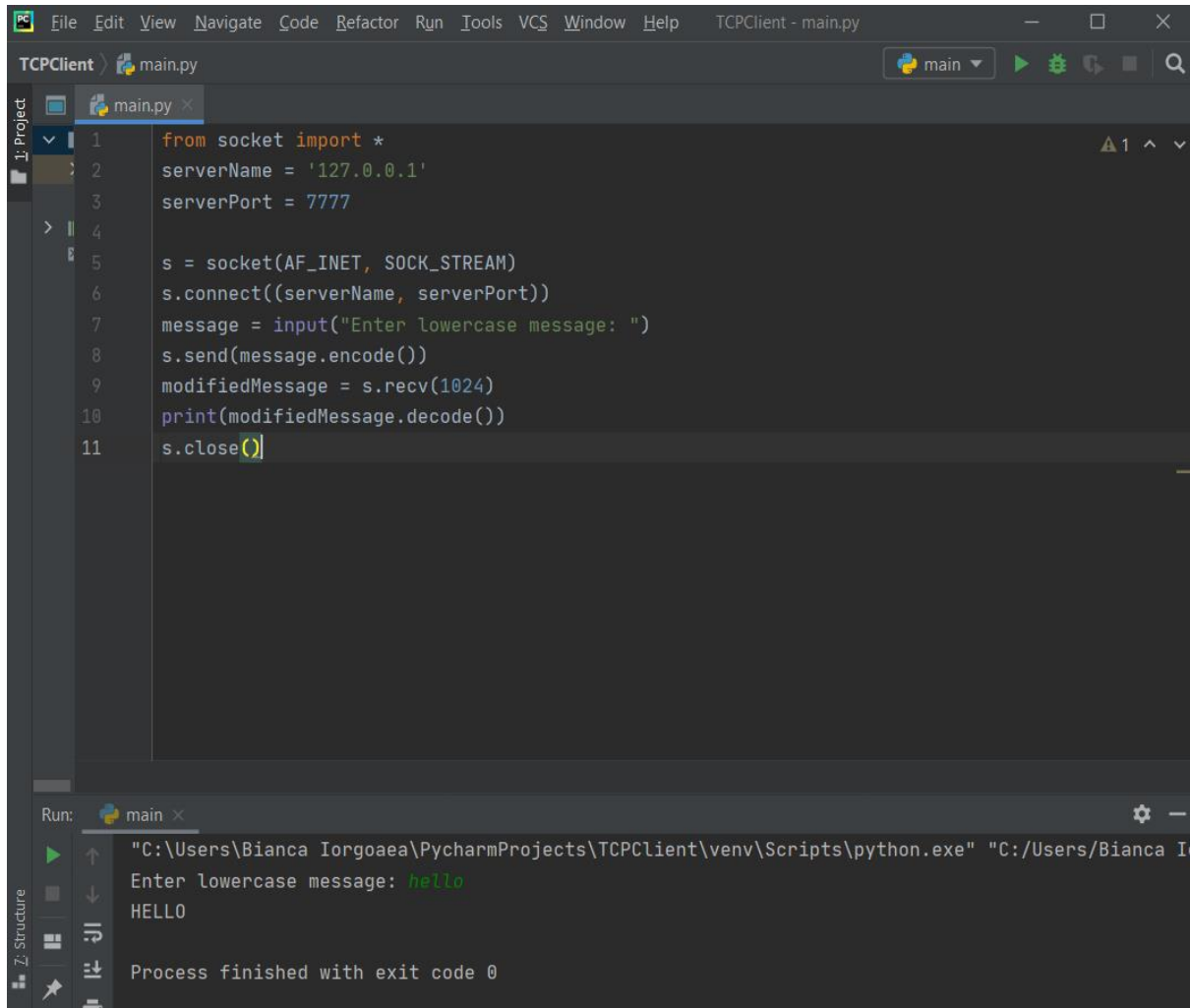
Python socket programming

TCP Server-Client

Înainte ca un client să poată trimite informație către un server, va avea nevoie să stabilească o conexiune între el și server prin intermediul *connect*, care conține adresa destinației. Socket-ul se conectează folosind adresa IP a serverului și un anumit port. Spre deosebire de UDP, mesajul transmis de către utilizator nu va fi explicit inclus într-un pachet care trebuie să ajungă la o anumită destinație specificată ca parametru, ci va trimite șirul de caractere prin intermediul variabilei *message*. Răspunsul din partea serverului va fi stocat într-o variabilă separată, *modifiedMessage*.

Conexiunea TCP va avea nevoie de transmiterea informației sub formă de bytes, nu de caractere, astfel că șirul citit de la tastatură trebuie mai întâi convertit din tipul string în tipul

byte înainte de a fi trimis către server. Același principiu se aplică și atunci când primim informația în tipul byte și trebuie s-o afișăm ca șir de caractere.



The screenshot displays the PyCharm IDE interface. The top toolbar includes icons for File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window shows a file named `main.py` with the following Python code:

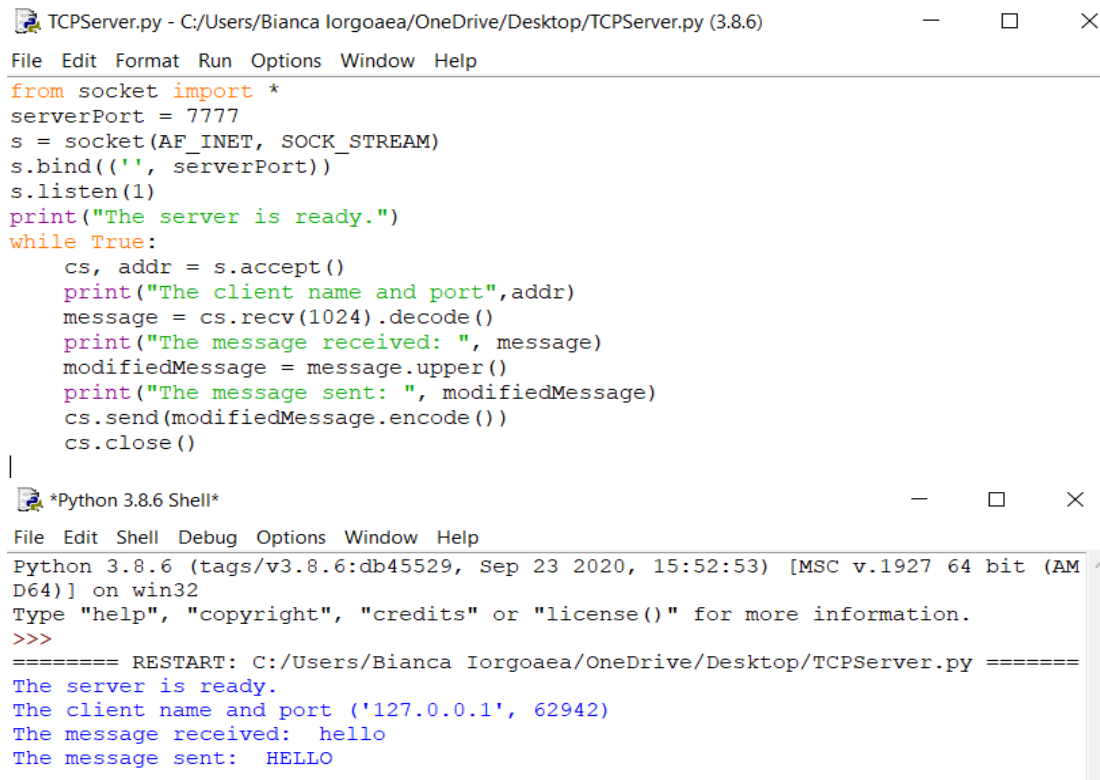
```
1 from socket import *
2 serverName = '127.0.0.1'
3 serverPort = 7777
4
5 s = socket(AF_INET, SOCK_STREAM)
6 s.connect((serverName, serverPort))
7 message = input("Enter lowercase message: ")
8 s.send(message.encode())
9 modifiedMessage = s.recv(1024)
10 print(modifiedMessage.decode())
11 s.close()
```

Below the editor, the Run console shows the execution of the program. The command prompt indicates the path to the Python interpreter. The input "hello" is shown in green, and the output "HELLO" is shown in black. The console also displays "Process finished with exit code 0".

Fig 2.1. TCP Client în Python

Socket-ului din cadrul serverului i se asociază un număr de port, astfel încât dacă un client va transmite date către portul 7777 al adresei IP specificate, atunci pachetul va fi direcționat direct către socket. Va aștepta ca un client să se conecteze și să trimită informația ce trebuie modificată, după ce a fost specificat, ca parametru al funcției *listen*, numărul maxim posibil de conexiuni – în cazul nostru, una. În momentul în care serverul primește o cerere de conexiune, se va crea un nou socket dedicat unui singur client, pe care l-am numit *cs* (connection socket). Se realizează o conexiune TCP între vechiul socket declarat la începutul programului și noul socket, ca mai apoi serverul și clientul să poată transmite reciproc informație. Fiind vorba de o conexiune TCP, se garantează că mesajele vor fi transmise în

ordine. La final, după ce serverul a primit mesajul și a răspuns cu varianta sa modificată, am închis socket-ul creat pentru client, însă altcineva va putea în continuare să se conecteze.



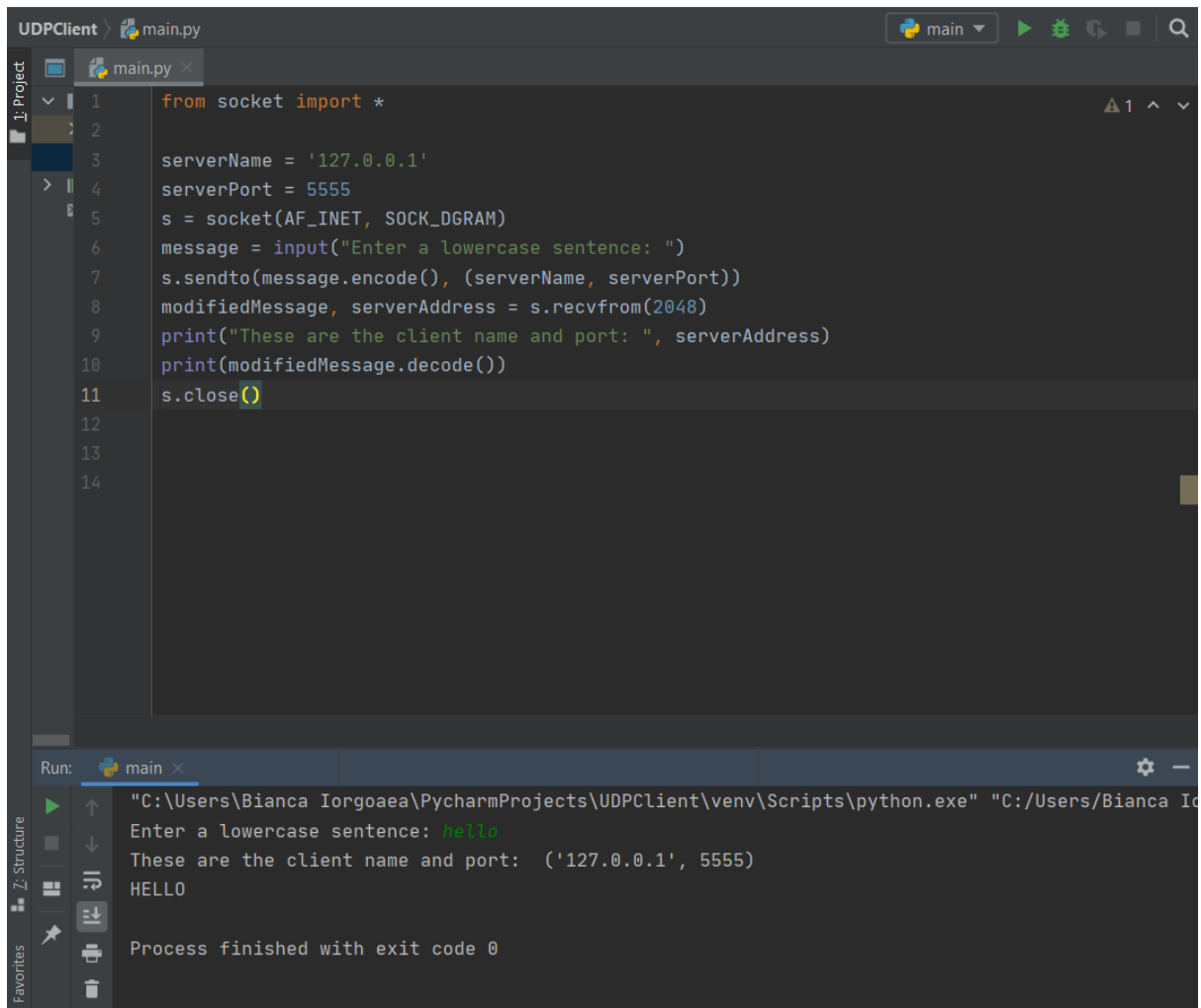
```
TCPServer.py - C:/Users/Bianca Iorgoaea/OneDrive/Desktop/TCPServer.py (3.8.6)
File Edit Format Run Options Window Help
from socket import *
serverPort = 7777
s = socket(AF_INET, SOCK_STREAM)
s.bind('', serverPort)
s.listen(1)
print("The server is ready.")
while True:
    cs, addr = s.accept()
    print("The client name and port",addr)
    message = cs.recv(1024).decode()
    print("The message received: ", message)
    modifiedMessage = message.upper()
    print("The message sent: ", modifiedMessage)
    cs.send(modifiedMessage.encode())
    cs.close()

*Python 3.8.6 Shell*
File Edit Shell Debug Options Window Help
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Bianca Iorgoaea/OneDrive/Desktop/TCPServer.py =====
The server is ready.
The client name and port ('127.0.0.1', 62942)
The message received: hello
The message sent: HELLO
```

Fig 2.2. TCP Server în Python

UDP Server-Client

Asemănător cu cazul în care am avut conexiune TCP, este nevoie să creăm un socket și să citim un mesaj de la utilizator, făcând transformarea din tipul string în tipul byte. Metoda *sendto()* asociază explicit mesajul cu adresa destinație cu ajutorul parametrilor, iar adresa sursei este implicită. Atunci când mesajul prelucrat ajunge la socket-ul clientului, va fi stocat într-o variabilă separată, primind și adresa serverului – variabila *serverAddress* va conține atât adresa IP a destinatarului, cât și numărul portului.



The screenshot displays the PyCharm IDE interface. The top pane shows the code for `main.py` in a project named `UDPClient`. The code is as follows:

```
1 from socket import *
2
3 serverName = '127.0.0.1'
4 serverPort = 5555
5 s = socket(AF_INET, SOCK_DGRAM)
6 message = input("Enter a lowercase sentence: ")
7 s.sendto(message.encode(), (serverName, serverPort))
8 modifiedMessage, serverAddress = s.recvfrom(2048)
9 print("These are the client name and port: ", serverAddress)
10 print(modifiedMessage.decode())
11 s.close()
```

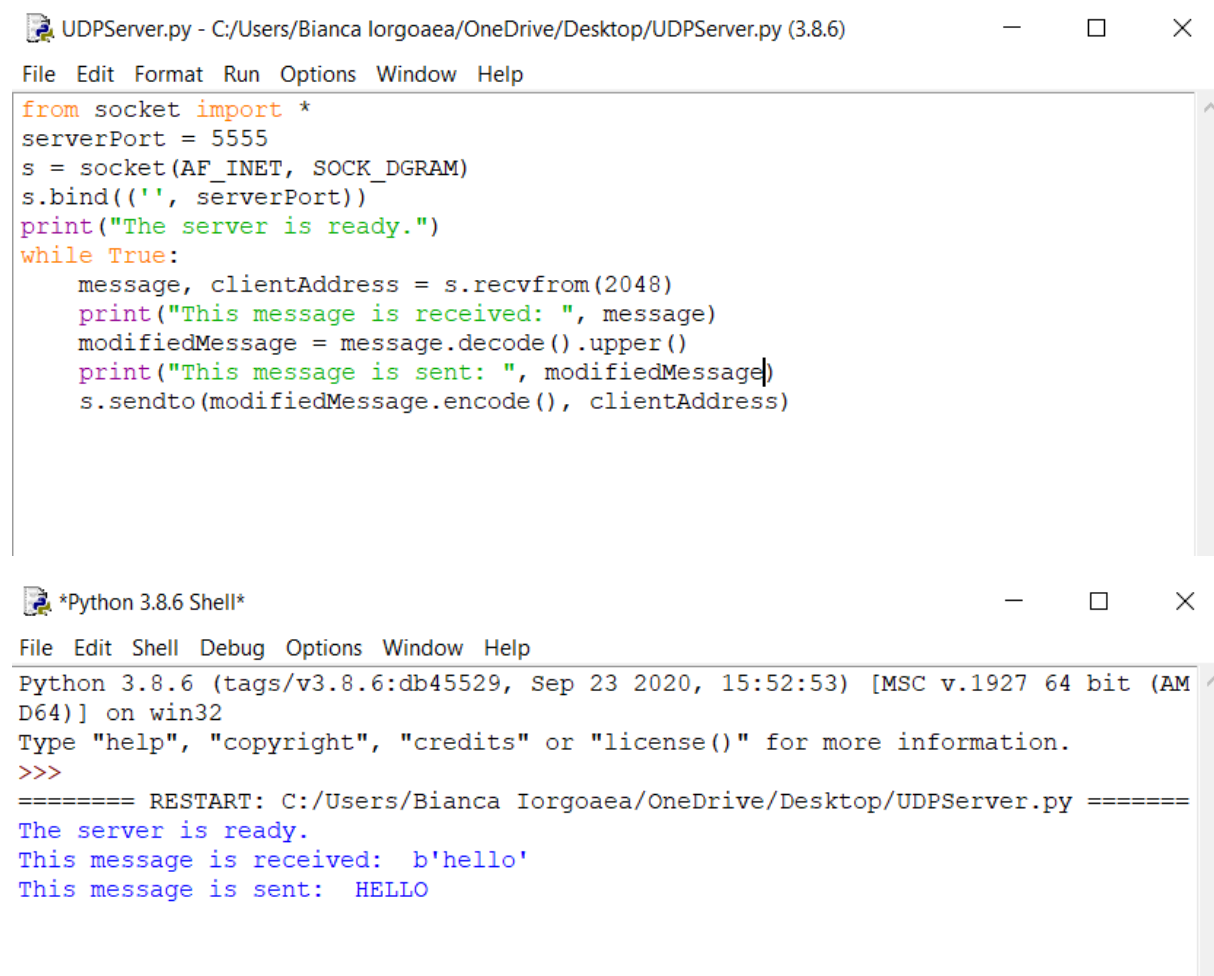
The bottom pane shows the Run console output for the `main` process:

```
"C:\Users\Bianca Iorgoaea\PycharmProjects\UDPClient\venv\Scripts\python.exe" "C:/Users/Bianca Iorgoaea/PycharmProjects/UDPClient/main.py"
Enter a lowercase sentence: hello
These are the client name and port: ('127.0.0.1', 5555)
HELLO
Process finished with exit code 0
```

Fig 2.2. UDP Client în Python

În cazul serverului, socket-ului i se va asocia de la început un număr de port, așa cum se întâmplă și în cazul conexiunii TCP. Serverul primește, precum clientul, atât mesajul, cât și

adresa sursei.



The image shows two windows from a Python IDE. The top window, titled 'UDPServer.py - C:/Users/Bianca Iorgoaea/OneDrive/Desktop/UDPServer.py (3.8.6)', contains the following Python code:

```
from socket import *
serverPort = 5555
s = socket(AF_INET, SOCK_DGRAM)
s.bind('', serverPort)
print("The server is ready.")
while True:
    message, clientAddress = s.recvfrom(2048)
    print("This message is received: ", message)
    modifiedMessage = message.decode().upper()
    print("This message is sent: ", modifiedMessage)
    s.sendto(modifiedMessage.encode(), clientAddress)
```

The bottom window, titled '*Python 3.8.6 Shell*', shows the execution output:

```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Bianca Iorgoaea/OneDrive/Desktop/UDPServer.py =====
The server is ready.
This message is received:  b'hello'
This message is sent:  HELLO
```

Fig 2.3 UDP Server în Python

TCP Server-Client, concurent/multithreaded

Conexiunea TCP implementată concurent se referă la faptul că serverul va fi capabil să gestioneze simultan mai mulți clienți, nu doar unul singur.

Implementarea clienților e identică cu implementarea unei conexiuni TCP obișnuite, ca în exemplele anterioare.

The screenshot displays the PyCharm IDE interface. The main editor window shows a Python script named `main.py` for a TCP client. The code is as follows:

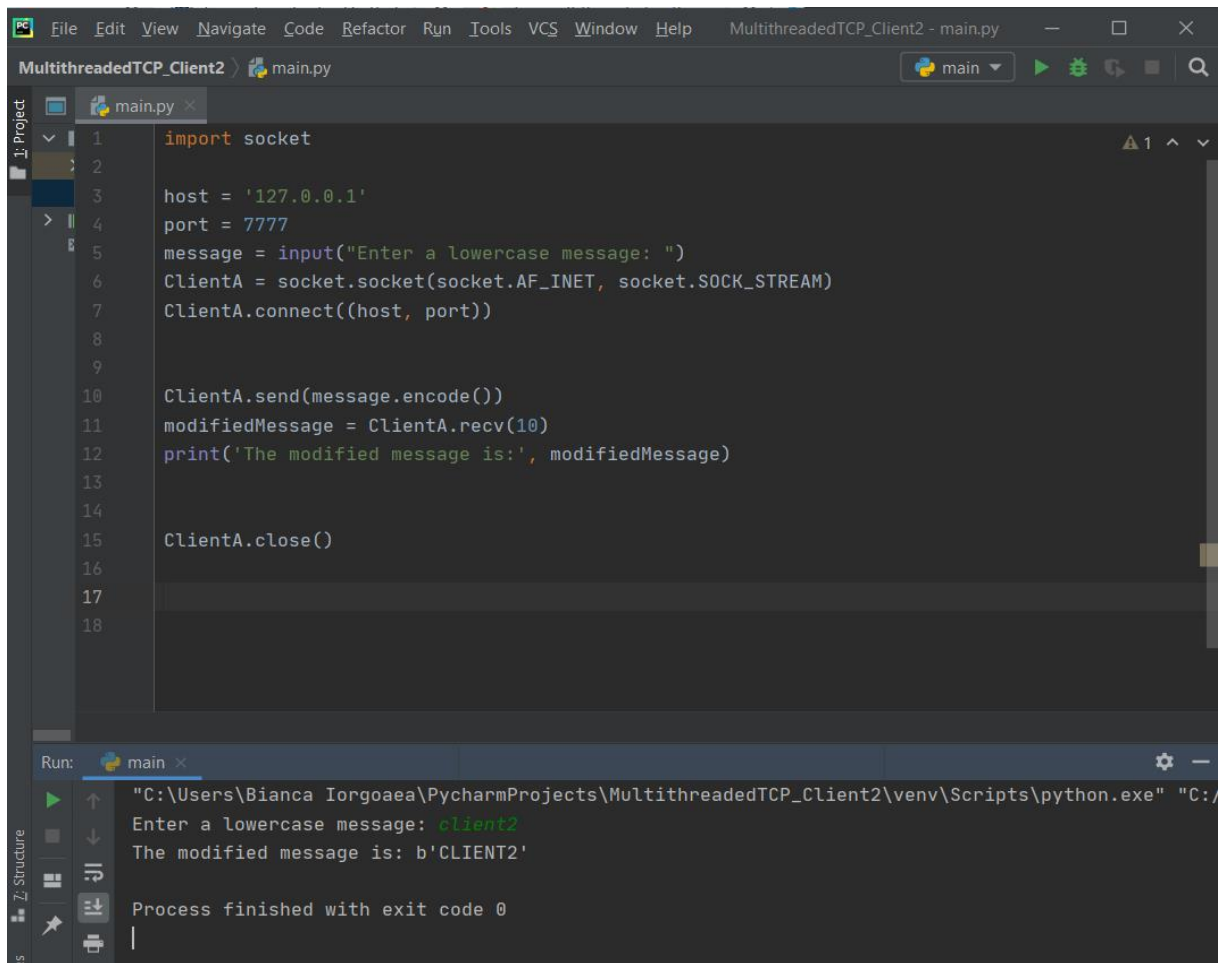
```
1 import socket
2
3 host = '127.0.0.1'
4 port = 7777
5 message = input("Enter a lowercase message: ")
6 ClientA = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 ClientA.connect((host, port))
8
9
10 ClientA.send(message.encode())
11 modifiedMessage = ClientA.recv(10)
12 print('The modified message is: ', modifiedMessage)
13
14
15 ClientA.close()
```

The bottom panel shows the Run console output for the `main` process:

```
Run: main x
"C:\Users\Bianca Iorgoaea\PycharmProjects\MultithreadedTCP\venv\Scripts\python.exe" "C:/Users
Enter a lowercase message: client1
The modified message is: b'CLIENT1'

Process finished with exit code 0
```

Fig 2.4 TCP concurrent în Python, Client A



The screenshot displays the PyCharm IDE interface. The main editor window shows a Python file named `main.py` with the following code:

```
1 import socket
2
3 host = '127.0.0.1'
4 port = 7777
5 message = input("Enter a lowercase message: ")
6 ClientA = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 ClientA.connect((host, port))
8
9
10 ClientA.send(message.encode())
11 modifiedMessage = ClientA.recv(10)
12 print('The modified message is:', modifiedMessage)
13
14
15 ClientA.close()
16
17
18
```

The bottom panel shows the Run console output for the `main` configuration:

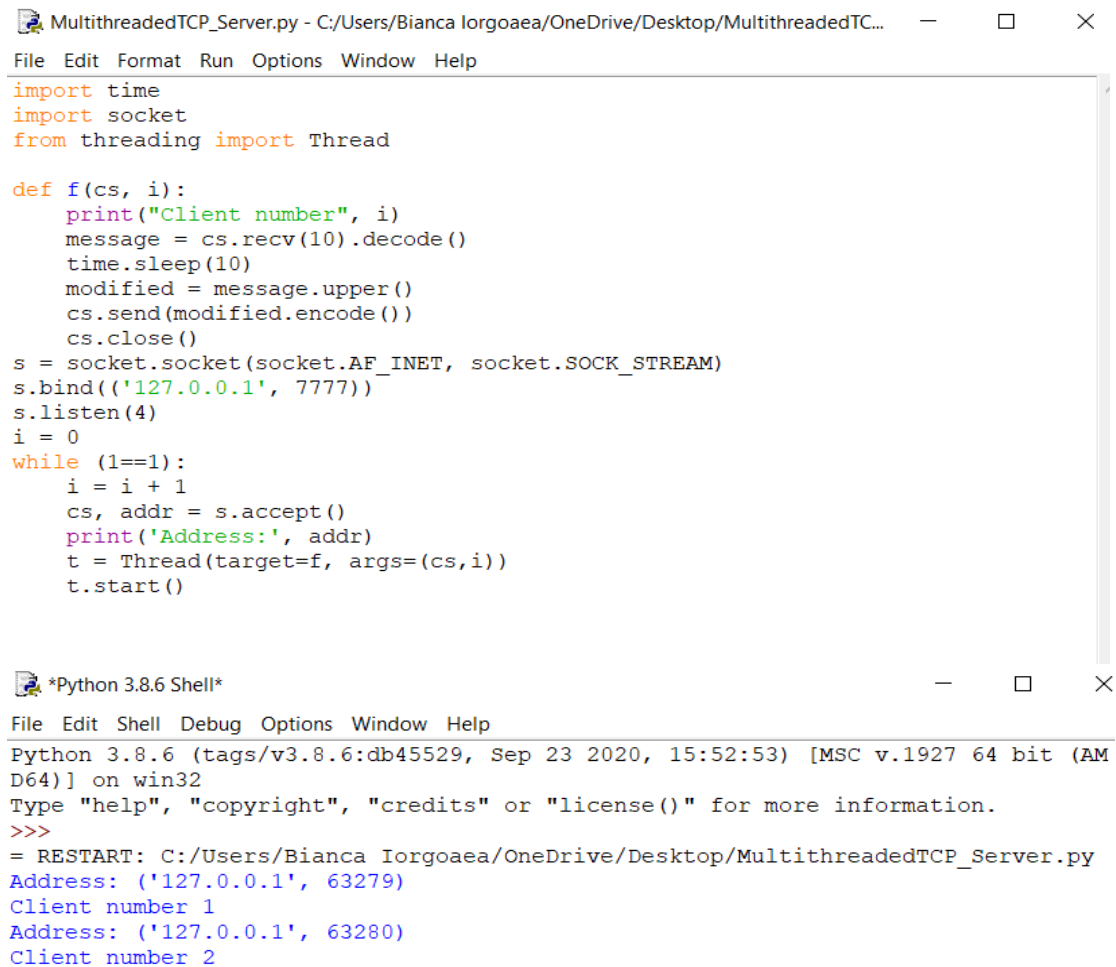
```
"C:\Users\Bianca Iorgoaea\PycharmProjects\MultithreadedTCP_Client2\venv\Scripts\python.exe" "C:/
Enter a lowercase message: client2
The modified message is: b'CLIENT2'

Process finished with exit code 0
```

Fig 2.5. TCP concurrent în Python, Client B

Metoda definită în cadrul serverului incrementează mai întâi numărul clienților conectați. Am folosit același *connection socket* dedicat unui client în mod particular, care primește datele de la client și îi transmite răspunsul cerut, urmând ca la finalul metodei să fie închisă conexiunea acestui socket. Fiind conexiune TCP, un al doilea client va putea să se conecteze la socket-ul serverului creat în afara metodei, iar procedura se repetă ca în cazul primului. În momentul în care unul dintre clienți transmite informație către socket-ul special al serverului, se apelează metoda definită.

Implementarea concurrentă nu presupune ca cei doi clienți să transmită informație simultan, dar clientul A și clientul B se pot conecta în același timp. Fiecare dintre cei doi clienți va primi, pe rând, mesajul modificat în majuscule.



The image shows a screenshot of a Python IDE window titled "MultithreadedTCP_Server.py - C:/Users/Bianca Iorgoaea/OneDrive/Desktop/MultithreadedTC...". The window contains the following Python code:

```
import time
import socket
from threading import Thread

def f(cs, i):
    print("Client number", i)
    message = cs.recv(10).decode()
    time.sleep(10)
    modified = message.upper()
    cs.send(modified.encode())
    cs.close()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 7777))
s.listen(4)
i = 0
while (1==1):
    i = i + 1
    cs, addr = s.accept()
    print('Address:', addr)
    t = Thread(target=f, args=(cs,i))
    t.start()
```

Below the code editor is a Python 3.8.6 Shell window titled "*Python 3.8.6 Shell*". It shows the execution of the script:

```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Bianca Iorgoaea/OneDrive/Desktop/MultithreadedTCP_Server.py
Address: ('127.0.0.1', 63279)
Client number 1
Address: ('127.0.0.1', 63280)
Client number 2
```

Fig 2.6. TCP Server concurrent în Python