

Tema UDP / TCP / TCP - Concurent

Nume: Muntean Ioan

Grupa: 722

Problema propusa:

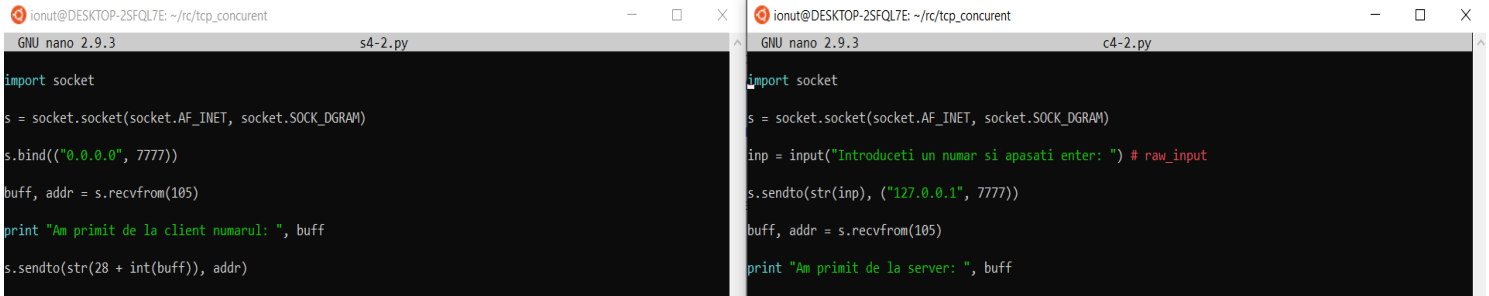
[Set probleme mai complicate - 4.2:](#)

Intoarce suma cifrelor din portul serverului adunate cu un numar primit de la client

UDP

Server

Client



```
ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
GNU nano 2.9.3 s4-2.py
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("0.0.0.0", 7777))
buff, addr = s.recvfrom(1024)
print "Am primit de la client numarul: ", buff
s.sendto(str(28 + int(buff)), addr)
```

```
ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
GNU nano 2.9.3 c4-2.py
import socket

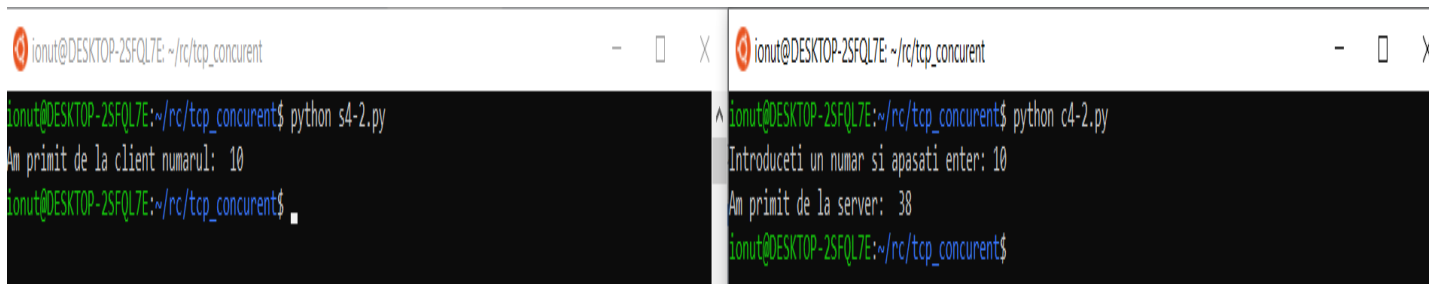
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
inp = input("Introduceti un numar si apasati enter: ") # raw_input
s.sendto(str(inp), ("127.0.0.1", 7777))
buff, addr = s.recvfrom(1024)
print "Am primit de la server: ", buff
```

In server importam libraria socket pentru a ne crea obiecte de tip socket. Cream un obiect de tip socket care primeste adrese IP din familia AF_INET si este de tip datagram socket. Apelam metoda bind pentru a stabili cine se poate conecta la acest server ("0.0.0.0" ne spune ca oricine se poate conecta) si care este portul server-ului la care sa se stabileasca conexiunea; in acest caz, portul l-am ales noi "7777", este arbitrar(**Atentie! Porturile de la 0 pana la 1023 sunt rezervate**).

Metoda recvfrom(nr_bytes) returneaza un tuplet de forma (informatie, adresa) unde adresa reprezinta adresa clientului care a trimis informatia catre server, iar nr_bytes reprezinta cat de mult sa se citeasca din canalul de comunicare.

Metoda sendto(buff, addr) va trimite un string buff la adresa mentionata addr (adresa retinuta mai sus, la apelul metodei recvfrom).

În client nu mai este necesar bind-ul; socket-urile de tip UDP nu stabilesc o conexiune client – server (spre deosebire de cele TCP, după cum vom vedea). Trebuie doar să cunoaștem ce trimitem și încotro trimitem. În cazul acesta, vom trimite la adresa ("127.0.0.1", 7777). 127.0.0.1 reprezintă adresa de loopback; semnifică localhost-ul. Iar portul 7777 este portul prestabilit. Mai departe, doar vom apela metodele `sendto(buff, addr)`, respectiv `recvfrom(nr_bytes)` - dacă este cazul.

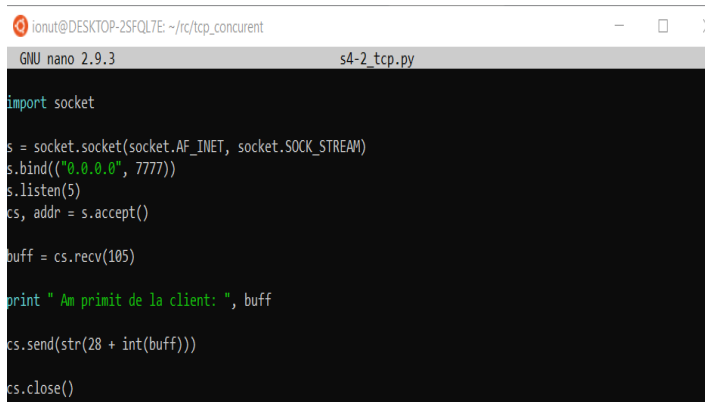


```
ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
ionut@DESKTOP-2SFQL7E:~/tcp_concurrent$ python s4-2.py
Am primit de la client numarul: 10
ionut@DESKTOP-2SFQL7E:~/tcp_concurrent$

ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
ionut@DESKTOP-2SFQL7E:~/tcp_concurrent$ python c4-2.py
Introduceti un numar si apasati enter: 10
Am primit de la server: 38
ionut@DESKTOP-2SFQL7E:~/tcp_concurrent$
```

TCP

Server



```
GNU nano 2.9.3 s4-2_tcp.py

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("0.0.0.0", 7777))
s.listen(5)
cs, addr = s.accept()

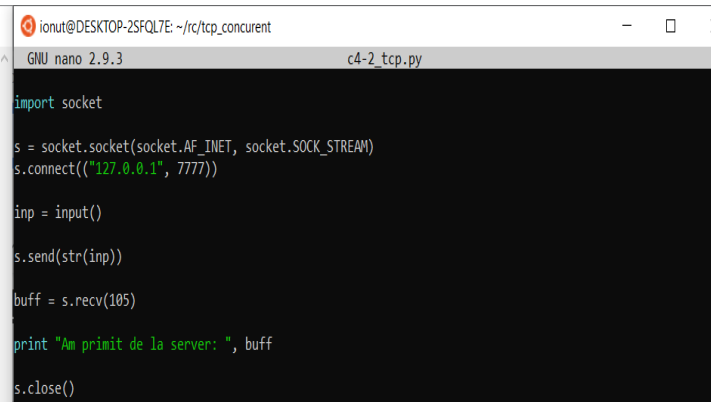
buff = cs.recv(105)

print "Am primit de la client: ", buff

cs.send(str(28 + int(buff)))

cs.close()
```

Client



```
GNU nano 2.9.3 c4-2_tcp.py

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1", 7777))

inp = input()

s.send(str(inp))

buff = s.recv(105)

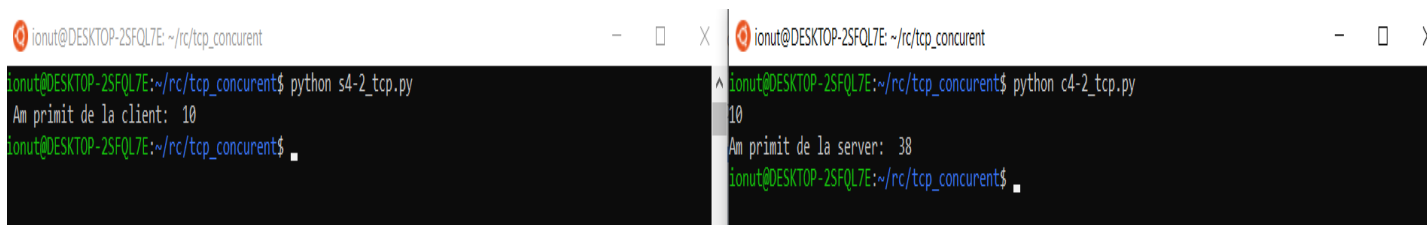
print "Am primit de la server: ", buff

s.close()
```

În cazul socket-urilor TCP, diferențele care apar sunt următoarele: TCP trebuie să stabilească o conexiune, să verifice erorile și să garanteze că fișierele sunt primite în ordinea în care au fost trimise; din aceste motive, această variantă este mai sigură, însă, în același timp, mai încetă.

In server, de aceasta data, al doilea parametru al metodei `socket.socket` este `SOCK_STREAM`. Apoi, continuam cu acelasi `bind` ca in prima varianta, insa mai departe vom apela `s.listen(5)` pentru a receptiona mai intai daca cineva scrie catre server. In continuare, retinem intr-un tuplet (`cs`, `addr`) - prin metoda `s.accept` -un “client socket” (prin care vom receptiona si trimite mesaje de la / catre client) respectiv adresa clientului. Comunicarea se va efectua de aceasta data cu `cs.send(nr_bytes)`, respectiv `cs.recv(nr_bytes)` (**Deci atentie: pentru comunicare, vom face apeluri de metode ale obiectului `cs`, returnat de `s.accept()`, iar nu ale obiectului `s`, creat initial!**). Nu mai trebuie sa stim catre cine trimitem sau de la cine primim, deoarece avem o conexiune stabilita de aceasta data. La final, inchidem socketul cu `s.close()` pentru a nu pierde date.

In client trebuie sa se stabileasca o conexiune. Nu vom trimite direct informatia , ci vom apela `s.connect` cu aceeasi parametri ca in cazul UDP. Din nou, primim & trimitem cu `s.send(nr_bytes)`, respectiv `s.recv(nr_bytes)`, iar la final inchidem socket-ul cu `s.close`



```
ionut@DESKTOP-2SFQL7E: ~/rc/tcp_concurrent
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python s4-2_tcp.py
Am primit de la client: 10
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$

ionut@DESKTOP-2SFQL7E: ~/rc/tcp_concurrent
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python c4-2_tcp.py
10
Am primit de la server: 38
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$
```

TCP - Concurrent

Server

```
ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
GNU nano 2.9.3 s4-2_conc.py
import socket
import time
from threading import Thread

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def f(cs, addr, i):
    print "Procesez clientul ", addr, " cu numarul de ordine ", i
    buff = cs.recv(105)
    # time.sleep(10)
    cs.send( str( 28 + int(buff) ) )
    print "Am terminat de procesat clientul ", i
    cs.close()

s.bind(("0.0.0.0", 7777))
s.listen(5)

i = 0
while True:
    i = i + 1
    cs, addr = s.accept()
    t = Thread(target = f, args=(cs, addr, i))
    t.start()
```

Client

```
ionut@DESKTOP-2SFQL7E: ~/tcp_concurrent
GNU nano 2.9.3 c4-2_conc.py
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1", 7777))

inp = input("Introduceti numarul de trimis server-ului:")

s.send(str(inp))

buff = s.recv(105)
print "Am primit de la server: ", buff

s.close()
```

O conexiune “TCP concurenta” deserveste mai multi clienti simultan.

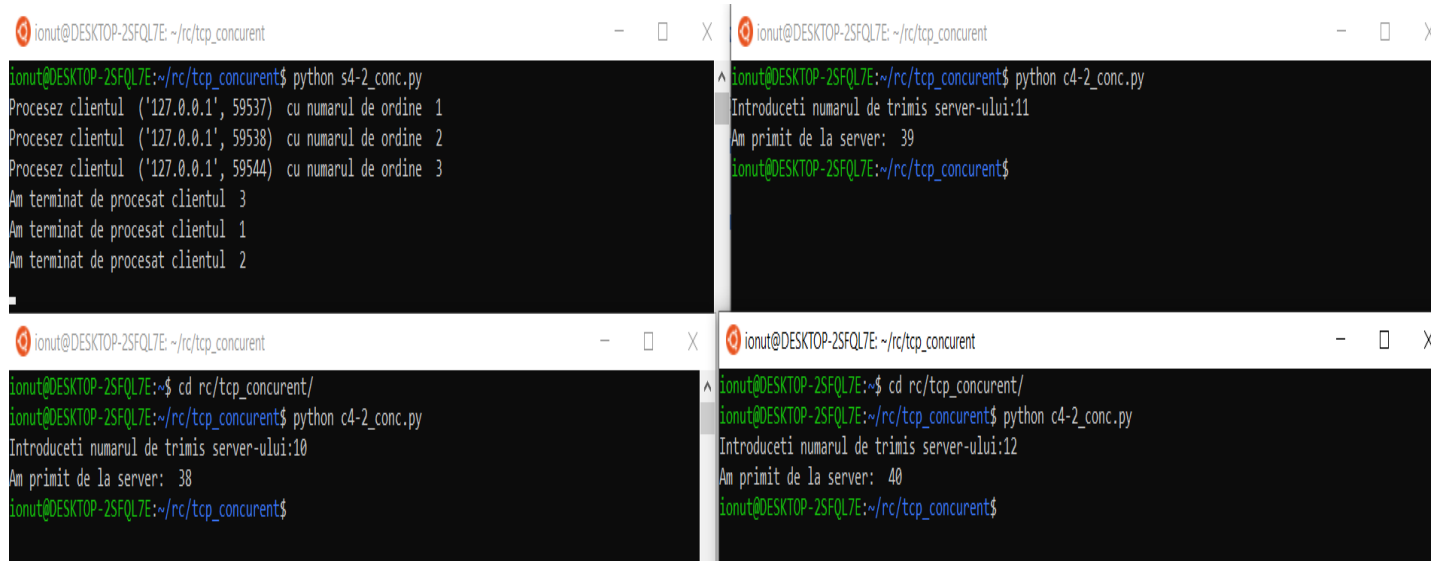
In client, spre deosebire de programul de mai sus cu TCP, vom avea o bucla infinita, in interiorul careia server-ul asteapta constant sa primeasca clienti, deci apelul `s.accept` se va muta in interiorul acestui loop. Apoi, de fiecare data cand un client face o cerere catre server, vom incepe un nou thread.

Ce este un thread? Este o modalitate prin care un program se poate imparti in doua sau mai multe sarcini care ruleaza simultan (sau pseudo-simultan). Thread-urile sunt mai ingaduitoare in ceea ce priveste resursele de sistem pe care le consuma, in comparatie cu procesele.

Un astfel de thread il vom crea de fiecare data cand vom avea un nou request de la un client, iar aici vom trata request-ul. Pentru aceasta avem nevoie de clasa ``Thread`` din biblioteca `threading`, deci o vom importa la inceput. In momentul in care ne cream un obiect de tip thread, ii vom pasa ca argumente constructorului o functie `f (“target = f”)` si o lista cu

argumentele pe care le ia aceasta functie ("args = (cs, addr, i)"). Rolul functiei este de a rezolva cererea si de a inchide client socket-ul la final.

Cat despre client, este identic cu un client TCP.



The image displays four terminal windows arranged in a 2x2 grid, illustrating the execution of a concurrent server and its clients. The top-left window shows the server script `s4-2_conc.py` processing three clients in order: 1, 2, and 3. The top-right window shows the client script `c4-2_conc.py` sending the number 11 to the server, which responds with 39. The bottom-left window shows the client script `c4-2_conc.py` sending the number 10 to the server, which responds with 38. The bottom-right window shows the client script `c4-2_conc.py` sending the number 12 to the server, which responds with 40. The windows are titled 'ionut@DESKTOP-2SFQL7E: ~/rc/tcp_concurrent'.

```
ionut@DESKTOP-2SFQL7E: ~/rc/tcp_concurrent
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python s4-2_conc.py
Procesez clientul ('127.0.0.1', 59537) cu numarul de ordine 1
Procesez clientul ('127.0.0.1', 59538) cu numarul de ordine 2
Procesez clientul ('127.0.0.1', 59544) cu numarul de ordine 3
Am terminat de procesat clientul 3
Am terminat de procesat clientul 1
Am terminat de procesat clientul 2

ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python c4-2_conc.py
Introduceti numarul de trimis server-ului:11
Am primit de la server: 39
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$

ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ cd rc/tcp_concurrent/
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python c4-2_conc.py
Introduceti numarul de trimis server-ului:10
Am primit de la server: 38
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$

ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ cd rc/tcp_concurrent/
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$ python c4-2_conc.py
Introduceti numarul de trimis server-ului:12
Am primit de la server: 40
ionut@DESKTOP-2SFQL7E:~/rc/tcp_concurrent$
```

Dupa cum se poate observa in imagine, am initializat 3 clienti intr-o ordine oarecare(in sens orar: sus-dreapta, jos-dreapta, jos-stanga), insa am introdus input-ul dupa care astepta fiecare in alta ordine (jos-stanga, sus-dreapta, jos-dreapta) si se poate observa ca serverul i-a tratat in ordinea in care s-a dat input-ul, deci concurrent.

Video: [Retele: problema 4.2 - UDP/TCP/TCP/Concurrent](#)