Proiect la Retele de Calculatoare

Aelenei Ciprian, seria C, grupa 1076

Aplicatia 1: Client TCP <-> Server TCP in limbajul de programare C

Socketurile TCP sunt utilizate pentru comunicarea dintre un server și un proces client. Codul serverului rulează mai întâi, care deschide un port și ascultă solicitările de conexiune primite de la clienți. Odată ce un client se conectează la același port (server), clientul sau serverul poate trimite un mesaj. Odată ce mesajul este trimis, oricine îl primește (server sau client) îl va procesa în consecință.

Următoarea aplicație client-server permite unui client să se conecteze la un server și să trimită un singur mesaj. Serverul răspunde cu *„Acesta este mesajul serverului”* și comunicarea se termină.

Explicatie partea de server:

1. Includeți fișierele antet sys/socket.hși arpa/inet.h: dsfghdsf

```
#include <sys/socket.h>
#include <arpa/inet.h>
```

2. Creați un socket care returnează un descriptor de socket; aceasta va fi utilizată pentru a face referire la socketul ulterior în cod:

```
int socket_desc = socket(AF_INET, SOCK_STREAM, 0);
```

Codul partea server păstrează informațiile de adresă atât ale serverului, cât și ale clientului într-o variabilă de tip sockaddr_in, care este struct.

3. Inițializați adresa serverului prin port și IP:

```
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(7602);
server_addr.sin_addr.s_addr = inet_addr("37.120.249.45");
```

4. Se face bind descriptorului de socket la adresa serverului:

```
bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr);
```

5. Porniți socketul pentru a asculta conexiunile primite:

```
listen(socket_desc, 1);
```

6. Stocați adresa clientului și descriptorul de socket acceptând o conexiune de intrare:

```
struct sockaddr client_addr;
int client_size = sizeof(client_addr);
int client_sock = accept(socket_desc, (struct sockaddr*)&client_addr,
&client_size);
```

Codul de pe partea serverului se oprește și așteaptă accept() până când primeste un client connect().

7. Comunicați cu clientul folosind send()și recv():

```
recv(client_sock, client_message, sizeof(client_message), 0);
send(client_sock, server_message, strlen(server_message), 0);
```

Când recv()este apelat, codul se oprește și așteaptă un mesaj de la client.

8. Închideți serverul și socketul clientului pentru a termina comunicarea:

```
close(client_sock);
close(socket_desc);
```

Explicatie partea de client:

1. Se includ fisiere antet, se implementeaza un socket si se initializeaza informațiile de adresă ale serverului într-o variabilă de tip sockaddr_in, similar cu modul în care a fost realizat la partea serverului:

```
#include <sys/socket.h>
#include <arpa/inet.h>

int socket_desc = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(2000);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

2. Trimiteți o cerere de conectare la server, care așteaptă la accept():

```
connect(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr));
```

3. Comunicați cu serverul folosind send() și recv():

```
send(socket_desc, client_message, strlen(client_message),0);
```

```
recv(socket_desc, server_message, sizeof(server_message),0);
```

Clientul așteaptă ca serverul să trimită un mesaj când recv() este apelat.

4. Închideți socketul:

```
close(socket_desc);
```

Un blocaj va apărea dacă atât clientul, cât și serverul așteaptă mesajul celuilalt la recv().

Aplicatia 2: Client TCP <-> Server TCP in Python

Python este un limbaj de programare excelent pentru rețelele de calculatoare. Ne permite să creăm aplicații solide foarte rapid și ușor. În acest exemplu, voi implementa un chat TCP. Voi avea un server care găzduiește chatul și mai mulți clienți care se conectează la acesta și comunică între ei.

Explicatie partea de server:

1. Pentru implementarea serverului, va trebui să importăm două biblioteci, și anume socket și threading. Prima va fi utilizata pentru conexiunea la rețea, iar cea de-a doua este necesară pentru îndeplinirea diferitelor sarcini în același timp.

```
import socket
import threading
```

2. Următoarea sarcină este să ne definim datele de conexiune și să ne inițializăm socketul. Vom avea nevoie de o adresă IP pentru gazdă și un număr de port liber pentru serverul nostru. În acest exemplu, voi folosi adresa ***37.120.249.45*** și portul atribuit de catre admin-ul serverului, ***7602***. Portul este de fapt irelevant, dar trebuie să vă asigurați că portul pe care îl utilizați este liber și nu este rezervat.

```
host = “37.120.249.45”
port = 7602
```

```
clients = []
nicknames = []
```

3. Când ne definim socket-ul, trebuie să trecem doi parametri. Acestea definesc tipul de socket pe care dorim să îl folosim. Primul (AF_INET) indică faptul că folosim un socket de internet mai degrabă decât un socket unix. Al doilea parametru reprezintă protocolul pe care dorim să îl folosim. SOCK_STREAM indică faptul că folosim TCP și nu UDP. După definirea socketului, îl legăm de gazda noastră și de portul specificat, trecând un tuplu care conține ambele valori. Apoi punem serverul nostru în modul ascultare, astfel încât să aștepte conectarea clienților. La final, creem două liste goale, pe care le vom folosi pentru a stoca clienții conectați și nickname-urile acestora.

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()

clients = []
nicknames = []
```

4. Aici definim o mică funcție care ne va ajuta să difuzăm mesaje și să facem mai ușor de citit codul. Ceea ce face este doar trimiterea unui mesaj către fiecare client conectat și, prin urmare, în lista de clienți. Vom folosi această metodă în celelalte metode.

```
def broadcast(message):
    for client in clients:
            client.send(message)
```

5. Acum vom începe cu implementarea primei funcții majore. Această funcție va fi responsabilă pentru gestionarea mesajelor de la clienți.

```
def handle(client):
    while True:
        try:

            message = client.recv(1024)
            broadcast(message)
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast('{} left!'.format(nickname).encode('ascii'))
            nicknames.remove(nickname)
                break
```

6. După cum puteți vedea, această funcție rulează într-o buclă. Nu se va opri decât dacă există o excepție din cauza unui lucru care este greșit. Funcția acceptă un client ca parametru. De fiecare dată când un client se conectează la serverul nostru executăm această funcție pentru acesta și astfel se pornește o buclă infinita care face să se

primească mesajul de la client (dacă acesta îl trimite) și să îl transmită tuturor clienților conectați. Deci, atunci când un client trimite un mesaj, toți ceilalți pot vedea acest mesaj. Acum, dacă din anumite motive există o eroare la conexiunea cu acest client, îl eliminăm, închidem conexiunea și difuzăm că acest client a părăsit chatul. După aceea rupem bucla și acest fir de executie se termină.

```
def receive():
    while True:
        client, address = server.accept()
        print("Connected with {}".format(str(address)))

        client.send('NICK'.encode('ascii'))
        nickname = client.recv(1024).decode('ascii')
        nicknames.append(nickname)
        clients.append(client)

        print("Nickname is {}".format(nickname))
        broadcast("{} joined!".format(nickname).encode('ascii'))
        client.send('Connected to server!'.encode('ascii'))

        thread = threading.Thread(target=handle, args=(client,))
              thread.start()
```

7. Când suntem gata să ne rulăm serverul, vom executa această funcție de primire. De asemenea, începe o buclă interminabilă care acceptă în mod constant noi conexiuni de la clienți. Odată ce un client este conectat, acesta îi trimite șirul „NICK”, care îi va spune clientului că porecla este solicitată. După aceea, așteaptă un răspuns (care, sperăm să conțină nicknameul) și adaugă clientul cu nicknameul respectiv în liste. După aceea, afișăm și transmitem aceste informații. În cele din urmă, începem un nou fir de execuție care rulează funcția de manipulare implementată anterior pentru acest client.

```
    receive()
```

Explicatie partea de client:

1. Importăm aceleași biblioteci exact ca la implementarea serverului:

```
import socket
import threading
```

2. Primii pași ai clientului sunt alegerea unui nickname și conectarea la serverul nostru. Va trebui să știm adresa exactă și portul pe care rulează serverul nostru. Implementăm funcția de conectare la serverul implementat mai sus.

```
nickname = input("Alege nicknameul: ")
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      client.connect(('37.120.249.45', 7602))
```

3. Acum, un client trebuie să aibă două fire de execuție care rulează în același timp. Primul fir va primi în mod constant date de la server, iar al doilea fir va trimite propriile noastre mesaje către server, așadar vom avea nevoie de două funcții aici. Implementăm funcția de recepție. Avem o buclă continuă care primește mesaje în mod constant și le afișează pe ecran. În cazul unei erori rupem bucla.

```
def receive():
    while True:
        try:
            message = client.recv(1024).decode('ascii')
            if message == 'NICK':
                client.send(nickname.encode('ascii'))
            else:
                print(message)
        except:
            print("An error occured!")
            client.close()
                  break
```

4. Implementăm funcția de trimitere a mesajelor.

```
def write():
    while True:
        message = '{}: {}'.format(nickname, input(''))
              client.send(message.encode('ascii'))
```

5. Funcția de scriere rulează într-o buclă nesfârșită care așteaptă întotdeauna o intrare de la utilizator. Odată ce primește, îi atașează nickname-ul și îl trimite la server. Ultimul lucru pe care trebuie să-l facem este să pornim două fire de execuție care rulează aceste două funcții.

```
receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write)
      write_thread.start()
```