Universitatea Tehnica a Moldovei
Facultatea Calculatoare Informatică și Microelectronică

Departamentul ISA

Raport

Lucrarea de laborator Nr. 7
la Programarea în rețea

Tema: Proiectarea și crearea aplicației client server

A elaborat st. gr. TI-144
Gorduz D.
A verificat lect. asist.
Ostapenco S.

Chișinău 2017

Obiective

Înțelegerea mecanismului de comunicare în rețea în prisma conceptului fundamental numit socket, studiul operațiilor primare ce compun un API bazat pe socket; obiectivul specific constă în elaborarea unei aplicații client-server și descrierea structurată a protocolului ce definește interacțiunea dintre componentele distribuite ale sistemului.

Scopul lucrării

Crearea unui sistem client-server, incluzând proiectarea protocolului de comunicare UDP

Link repozitoriu distant GIT: https://github.com/GorduzDaniel/lab7_PR

Noțiuni teoretice

Serverul este o aplicație care ofera servicii clientilor sositi prin retea. Serverele ofera o gama variata de servicii. Serverul cel mai cunoscut este serverul Web, care furnizeaza documentele cerute de catre clienti. Un alt serviciu cunoscut este posta electronica, care utilizeaza protocoalele *SMPT* (Simple Mail Transfer Protocol) si *IMAP4* (Internet Mail Access Protocol). Pe principiul client-server funnctioneaza si protocoalele *NFS* (Network File Service) si *FTP* sau serviciul de nume de domenii *DNS* (Domain Name Service) si serviciul de informatii despre retea *NIS* (Network Information Services). Trebuie sa amintim si serviciul care permite logarea la un calculator aflat la distanta: TELNET si RLOGIN. Putem trage concluzia ca arhitectura client-server este instrumentul de baza in dezvoltarea aplicatiilor de retea.

Clientul este o aplicatie care utilizeaza serviciile oferite de catre un server. Pentru a putea realiza acest lucru, clientul trebuie sa cunoasca unde se afla serverul in retea si cum trebuie comunicat cu acesta si ce servicii oferă. Deci dacă un client doreste o comunicare cu serverul, trebuie sa cunosca trei lucruri:

- adresa server
- portul server utilizat pentru comunicare
- protocolul de comunicatie utilizat de server

Un *soclu* este de fapt un nod abstract de comunicatie. Soclurile reprezinta o interfata de nivel scazut pentru comunicarea in retea. Soclurile permit comunicarea intre procese aflate pe acelasi calculator sau pe calculatoare diferite din retea. Mecanismul de socluri a fost definit prima data in BSD UNIX. Java suporta trei tipuri de socluri. Clasa Socket utilizeaza un protocol orientat pe conexiune (TCP), clasa DatagramSocket utilizeaza protocolul UDP la nivelul transport, care este un protocol neorientat pe conexiune. O alta varianta a DatagramSocket este MulticastSocket

utilizat pentru a trimite date deodata la mai multi receptori. Soclurile utilizeaza fluxuri de date (streamuri) pentru a trimite si a receptiona mesaje.[1]

Protocolul **UDP** lucreaza diferit de TCP. Atunci cind transmitem data prin TCP intii creem o conexiune. Cind conexiunea este stabilita protocolul TCO garanteaza ca datele vor ajunge la cealalta parte a conexiunii, ori va anunta de o eroare.

Protocolul Datagramelor Utilizator este un protocol de comunicatie pentru calculatoarele ce apartin nivelului transport (nivelul 4) al modelului standard OSI.

Împreună cu Protocolul de Internet (IP) , acesta face posibila livrarea mesajelor intr-o retea. Spre deosebire de protocolul TCP, UDP constituie modul de comunicatie fara conexiune. Este similar cu sistemul postal, in sensul ca pachetele de informatii sunt trimise in general fara confirmare de primire, in speranta ca ele vor ajunge, fara a exista o legatura efectiva intre expeditor si destinatar. Practic **UDP** este un protocol ce nu ofera siguranta sosirii datelor la destinatie (nu dispune de mecanisme de confirmare); totodata nu dispune nici de mecanisme de verificare a ordinii de sosire a datagramelor. UDP dispune, totusi, in formatul datagramelor, de sume de control pentru verificarea integritatii datelor sau de informatii privind numarul portului pentru adresarea diferitelor functii la sursa/destinatie.

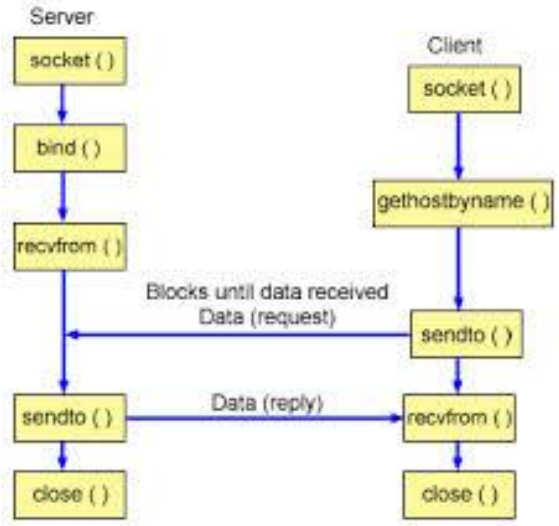

Figura 1 – Procesul de comunicare UDP

Protocolul UDP trimite pachete independente de date, numite **datagrame,** de la un calculator catre altul fara a garanta in vre-un fel ajungerea acestora la destinatie. Ca și segmentul TCP, segmentul UDP are la rindul sau un antet, ca in figura 2.

Figura 2- segmentul UDP

- Portul sursa – numarul de port al celui care face apelul;
- Portul destinatie – numarul portului apelat;
- Lungime – lungimea antetului si a datelor UDP;
- Suma de verificare (checksum) – suma cimpurilor de antet si date, calculata pentru verificare;
- Date – datele protocolului nivelului superior;

Protocolul UDP aduce in plus fata de IP numarul portului sursa si numarul portului destinatie. O datagrama UDP este plasata, impreuna cu portul sursa si portul destinatie, intr-un pachet IP, iar in cimpul protocol al pachetului IP se pune valoarea UDP.

UDP nu imparte fisierele are trebuiesc transmise in retea in parti mici; mesajului nesegmentat ii ataseza doua numere, reprezentind programul care trimite segmentul si cel care il primeste; nu asteapta confirmarea de primire din partea calculatorului destinatie si nu efectueaza transmisii. Poate fi considerat un TCP simplificat, mai rapid dar mai sigur.

Mersul lucrării

Pentru început în partea server, se creează un obiect de tip *DatagramSocket*, în constructorul căruia îi specificăm portul pe care va lucra serverul propriu-zis. Într-un ciclu infinit se include ca obiectul *DatagramSocket* să accepte conexiuni. În cazul când a avut loc o conexiune, pentru partea client conectată. Astfel în constructor se primește obiectul de tip Socket, care va servi pentru întreschimbarea de date între client și server. Acest lucru are loc cu ajutorul fluxurilor între clase cu ajutorul obiectelor *BufferedReader* În figura 1 este prezentată clasa Server.

```
import java.io.*;
import java.net.*;

public class Server{
    public static void main(String args[]){
        DatagramSocket sock = null;

        try{
            //1. creating a server socket, parameter is local port number
            sock = new DatagramSocket(7777);

            //buffer to receive incoming data
            byte[] buffer = new byte[65536];
            DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);

            //2. Wait for an incoming data
            echo("Server socket created. Waiting for incoming data...");

            //communication loop
            while(true)
            {
                sock.receive(incoming);
                byte[] data = incoming.getData();
                String s = new String(data, 0, incoming.getLength());

                //echo the details of incoming data - client ip : client port - client
message
                echo(incoming.getAddress().getHostAddress() + " : " +
incoming.getPort() + " - " + s);

                s = "OK : " + s;
                DatagramPacket dp = new DatagramPacket(s.getBytes() ,
s.getBytes().length , incoming.getAddress() , incoming.getPort());
                sock.send(dp);
            }
        }

        catch(IOException e)
        {
            System.err.println("IOException " + e);
        }
    }

    //simple function to echo data to terminal
    public static void echo(String msg)
    {
        System.out.println(msg);
    }
}
```

Figura 1 – Clasa server

În figura 3 este prezentată partea client a aplicației. Se creează un obiect de tip InetAdress în care se va extrage adresa hostului local. Și desigur pentru citirea și înscrierea în conexiune se fac obiecte de tip *BufferedReader* și *PrintWriter*, ca și pentru partea Server. La fel se creează un obiect Socket, în care îi este trimisă adresa la localhost, dar și portul serverului, creat anterior.

La fel se creează un ciclu while, în care condiția de ieșire va fi cuvântul cheie *quit*.

```
import java.io.*;
import java.net.*;

public class Client{
    public static void main(String args[]){
        DatagramSocket sock = null;
        int port = 7777;
        String s;

        BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));

        try{
            sock = new DatagramSocket();

            InetAddress host = InetAddress.getByName("localhost");

            while(true){
                //take input and send the packet
                echo("Enter message to send : ");
                s = (String)cin.readLine();
                byte[] b = s.getBytes();

                DatagramPacket  dp = new DatagramPacket(b , b.length , host , port);
                sock.send(dp);

                //now receive reply
                //buffer to receive incoming data
                byte[] buffer = new byte[65536];
                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                sock.receive(reply);

                byte[] data = reply.getData();
                s = new String(data, 0, reply.getLength());

                //echo the details of incoming data - client ip : client port - client
message
                echo(reply.getAddress().getHostAddress() + " : " + reply.getPort() + "
- " + s);
            }
        }

        catch(IOException e)
        {
            System.err.println("IOException " + e);
        }
    }

    //simple function to echo data to terminal
    public static void echo(String msg)
    {
        System.out.println(msg);
    }
}
```

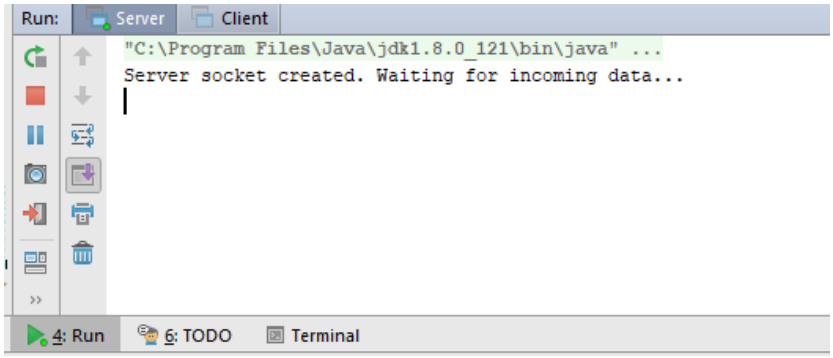

Figura 1 Conectarea la Server

Serverul a fost creat si asteapta date de la client.

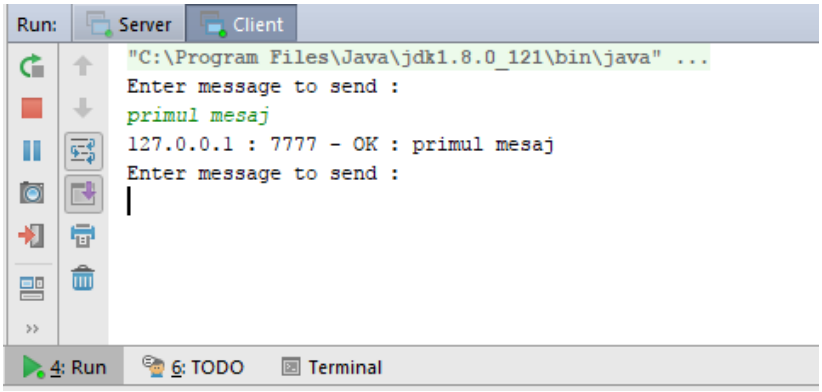

Figura 2 Clientul

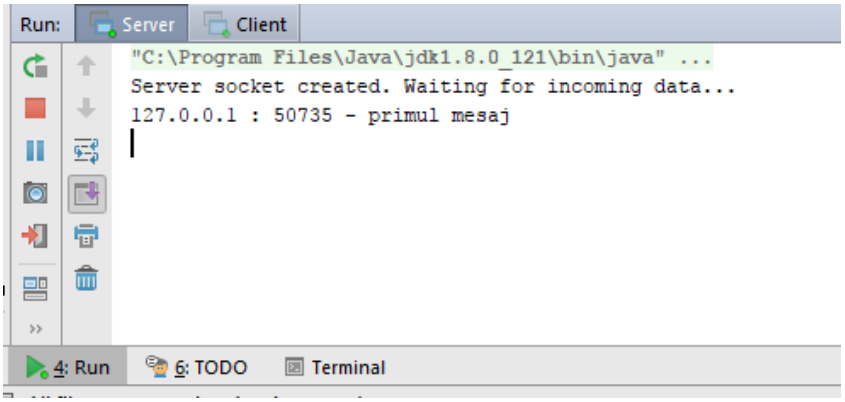

Figura 3 .Serverul primeste mesajul

Concluzie

În cadrul acestei lucrări de laborator au fost aprofundate cunoștințele în mecanismul de comunicare în rețea, și anume prin socket. Au fost obținute abilități în elaborarea unei aplicații multi-client – server, proiectându-se un simplu protocol de comunicare dintre ambii membri ai așa zisului dialog.

S-a observat că defapt comunicarea este o interacțiune orientată pe mesaje textuale și presupune defapt procesarea răspunsului. Datorită acestui fapt au fost aprofundate cunoștințele în limbajul de programare Java în cadrul comunicării de nivel jos în rețea.

Datorită acestei lucrări de laborator s-a înțeles principiul general, dar și cerințele pentru interschimbarea de informație în rețea a protocolului UDP.

Unul din avantajele ale protocolului UDP este ca pentru transmiterea a unei cantități mari de pachete nu are nevoie să creeze o conexiune, transmiterea datelor implică mai o întîrziere mai mică. Această întîrziere mică face UDP o alegere sigură pentru aplicațiile sensibile la delay ca și audio și video. Un alt avantaj ar fi aplicațiile multicast create deasupra la UDP întrucât el trebuie să transmită de la un point la multipoint. Folosind TCP pentur multicast aplicații va fi mai greu deoarece sender-ul ar trebui să ducă contul la rata retransmiterei/transmiterei pentru multipli receptori.

Bibliografie

1. Aplicații de rețea [Resursă electronică]. Regim de acces: https://www.ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs9/curs9.html

2. Tutorials Point [Resursă electronică]. Regim de acces: https://www.tutorialspoint.com/java/java_networking.htm