Laborator 3

**Sockets folosind UDP**

Obiectivele laboratorului:

* Laboratorul isi propune sa ilustreze prin exemple folosirea protocolului UDP pentru programarea in retele de calculatoare;
* Detalierea claselor *DatagramSocket, DatagramPacket;*
* Crearea unui server si a unui client folosind clasele de mai sus pentru a arata cum se face comunicarea.

Spre deosebire de socketurile de tip TCP/IP, socketurile pe baza de datagrame de tip UDP sunt de tip conectionless.

***Conectionless*** – conexiunea dintre client si server nu este mentinuta pe parcursul intregului dialog. Fiecare pachet este trimis ca o transmisiune izolata atunci cand este necesar. Socketurile de tip UDP sunt mult mai rapide in trimiterea de date decat socketurile TCP/IP dar nu sunt fiabile.

*Crearea serverului folosind datagrame UDP*

1. Cream un obiect de tip *DatagramSocket*

DatagramSocket datagramSocket = new DatagramSocket(port);

1. Cream un buffer pentru mesajele (datagramele) primite

byte[] buffer = new byte[256];

1. Cream un obiect de tip *DatagramPacket* pentru mesajele (datagramele) primite. Constructorul clasei pentru acest obiect are nevoie de doua argumente: bufferul creat anterior si dimensiunea acestuia.

DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);

1. Acceptam mesajul (datagrama) care a fost trimis de catre client. Acest lucru se realizeaza cu ajutorul metodei *receive()* a clasei *DatagramSocket* folosind obiectul *dp* de tip *DatagramPacket.*

datagramSocket.receive(inPacket);

1. Acceptam adresa si portul expeditorului din packet.

InetAddress adresaClient = inPacket.getAddress();

int portClient = inPacket.getPort();

1. Extragerea datelor din buffer. Datele sunt extrase ca un sir (string) folosind o forma suprascrisa a constructorului *String* care va avea nevoie de trei parametrii: un vector de tip byte ( = 0), pozitia de start a vectorului si numarul de biti ( = intreaga dimensiune a bufferului).

String mesaj = new String(inPacket.getData(), 0, inPacket.getLength());

1. Creeam datagramul (mesajul) folosit pentru raspuns. Acest lucru il facem prin crearea unui obiect de tip *DatagramPacket*, folosind o forma suprascrisa a constructorului care va avea nevoie de patru argumente: vectorul de tip byte care va contine mesajul de raspuns, dimensiunea mesajului de raspuns, adresa clientului, portul clientului.

Primul argument este returnat de metoda *getBytes* a clasei *String.*

DatagramPocket outPacket = new DatagramPacket(response.getBytes(), response.length(), clientAddress, clientPort);

In exemplul de sus, variabila *response* este o variabila de tip *String* care va contine mesajul returnat.

1. Trimiterea raspunsului (datagramei). Acest lucru este realizat prin invocarea metodei *send* a obiectului de tip *DatagramSocket,* furnizand ca parametru obiectul de tip *DatagramPacket* folosit pentru trimiterea mesajului ca argument.

datagramSocket.send(outPacket);

Pasii 4-8 se pot executa la infinit.

In circumstante normale, serverul nu este inchis definitiv. Oarecum, daca o exceptie are loc, atunci obiectul *DatagramSocket* asociat ar trebui inchis (a se vedea pasul 9).

1. Inchiderea obiectului *DatagramSocket.* Acest lucru se realizeaza folosind metoda *close* a clasei *DatagramSocket.*

datagramSocket.close();

De observat faptul ca exista doua diferente fata de metoda de creare a socketurilor folosind TCP/IP, si anume: **exceptia *IOException* din *main* este inlocuita de exceptia *SocketException***si **nu exista o exceptie generata de catre metoda *close* in clauza *finnally,* in concluzie nu exista blocul try...catch.**

Codul pentru server este urmatorul:

import java.io.\*;

import java.net.\*;

public class UDPEchoServer

{

private static final int PORT = 1234;

private static DatagramSocket datagramSocket;

private static DatagramPacket inPacket, outPacket;

private static byte[] buffer;

public static void main(String[] args)

{

System.out.println("Deschidere port...\n");

try

{

datagramSocket = new DatagramSocket(PORT);//Step 1.

}

catch(SocketException sockEx)

{

System.out.println("Imposibil de conectat la port!");

System.exit(1);

}

handleClient();

}

private static void handleClient()

{

try

{

String messageIn,messageOut;

int numMessages = 0;

do

{

buffer = new byte[256]; //Step 2.

inPacket =

new DatagramPacket(

buffer, buffer.length); //Step 3.

datagramSocket.receive(inPacket); //Step 4.

InetAddress clientAddress =

inPacket.getAddress(); //Step 5.

int clientPort =

inPacket.getPort(); //Step 5.

messageIn =

new String(inPacket.getData(),0,

inPacket.getLength()); //Step 6.

System.out.println("Mesaj primit.");

numMessages++;

messageOut = "Mesaje " + numMessages

+ ": " + messageIn;

outPacket = new DatagramPacket(

messageOut.getBytes(),

messageOut.length(),

clientAddress,

clientPort); //Step 7.

datagramSocket.send(outPacket); //Step 8.

}while (true);

}

catch(IOException ioEx)

{

ioEx.printStackTrace();

}

finally //Daca exceptia este aruncata, inchidem conexiune.

{

System.out.println("\n\* Inchidere conexiune... \*");

datagramSocket.close(); //Step 9.

}

}

}

*Crearea clientului folosind datagrame UDP*

1. Cream un obiect de tip *DatagramSocket*. Acest lucru este similar ca si mai sus, cu diferenta ca constructorul nu are nevoie de argumente, deoarece un port stabilit implicit (pe partea de client) este folosit.

DatagramSocket ds = new DatagramSocket();

1. Cream un mesaj (datagram) de iesire. Pasul este identic cu pasul 7 de mai sus.

DatagramPacket outPacket = new DatagramPacket(message.getBytes(), message.length(), host, PORT);

1. Trimiterea mesajului (datagram). Ca si pentru server, acest lucru este realizat pe client prin invocarea metodei *send* a clasei *DatagramSocket,* furnizand obiectul mesajului de iesire de tip *DatagramPacket* ca argument.

datagramSocket.send(outPacket);

Pasii 4-6 care urmeaza sunt similari (identici chiar) cu pasii 2-4 de pe server.

1. Creeam un buffer pentru datagramele (mesajele) de intrare.

byte[] buffer = new byte[256];

1. Creeam un obiect de tip DatagramPacket pentru mesajele (datagramele) primite.

DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);

1. Acceptare datagrame (mesaje) de intrare

datagramSocket.receive(inPacket);

1. Extragerea datelor din buffer

String response = new String(inPacket.length(),0,inPacket.getLength());

Pasii 2-7 se poate repeta de fiecare data cand este nevoie.

1. Inchiderea obiectului *DatagramSocket*

datagramSocket.Close();

Codul pentru client este urmatorul:

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class UDPEchoClient

{

private static InetAddress host;

private static final int PORT = 1234;

private static DatagramSocket datagramSocket;

private static DatagramPacket inPacket, outPacket;

private static byte[] buffer;

public static void main(String[] args)

{

try

{

host = InetAddress.getLocalHost();

}

catch(UnknownHostException uhEx)

{

System.out.println("Host ID not found!");

System.exit(1);

}

accessServer();

}

private static void accessServer()

{

try

{

datagramSocket = new DatagramSocket(); //Step 1.

//Set up stream for keyboard entry...

Scanner userEntry = new Scanner(System.in);

String message="", response="";

do

{

System.out.print("Enter message: ");

message = userEntry.nextLine();

if (!message.equals("\*\*\*CLOSE\*\*\*"))

{

outPacket = new DatagramPacket(

message.getBytes(),

message.length(),

host,PORT);

//Step 2.

datagramSocket.send(outPacket); //Step 3.

buffer = new byte[256]; //Step 4.

inPacket =

new DatagramPacket(

buffer, buffer.length); //Step 5.

datagramSocket.receive(inPacket); //Step 6.

response =

new String(inPacket.getData(),

0, inPacket.getLength());//Step 7.

System.out.println(

"\nSERVER> " + response);

}

}while (!message.equals("\*\*\*CLOSE\*\*\*"));

}

catch(IOException ioEx)

{

ioEx.printStackTrace();

}

finally

{

System.out.println("\n\* Closing connection... \*");

datagramSocket.close(); //Step 8.

}

}

}