**Ministerul Educaţiei al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Raport**

Lucrare de laborator **№6**

Disciplina: Programarea in Retea

A îndeplinit: st.gr.FI-161 Panfil Nicolae

A controlat: Gavrilita Mihai

Chișinău 2019

**Lucrare de Laborator Nr.6**

**Scop:**

**1. Ințelegerea protocolului UDP și metodele de bază;**

**2. Inițializarea proprietăților pentru protocolul UDP;**

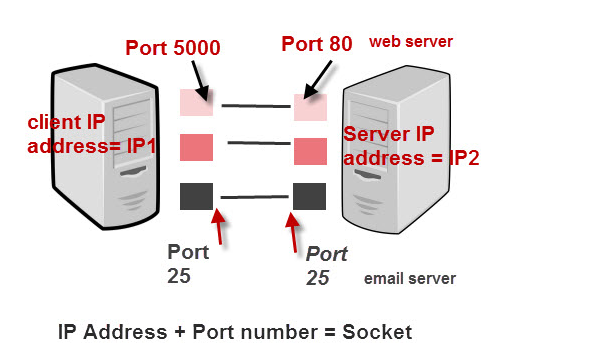
**3. Efectuarea unui chat pe baza conexiunii protocolului UDP;**

**4. Concepte de bază ale protocolului UDP.**

**Sarcina:**

Efectuarea aplicației Client-Server bazată pe conexiunea protocolului UDP.

Incepand cu faptul ca IP + Port Number = Socket, atunci un socket care este utilizat pentru efectuarea conexiunilor dintre Client și Server este punctul final al comunicării bidirecționale între două mașini care rulează în rețea cunoscând portul gazdă, ceea ce înseamnă pentru TCP că poate identifica aplicația și datele care trebuie trimise.

[](https://user-images.githubusercontent.com/43058513/56795778-d9db2680-6819-11e9-89e1-32aa6b1ad6cc.png)

Socket-ul asigură, în cazul dat, comunicarea între Client și Server. Pentru partea Client ar trebui să specific IP și Port, Iar pentru partea Server ar trebui să specific doar Portul. Socket-ul, cu alte cuvinte, deschide o ușă, deschide o cale de comunicare.

Protocolul UDP are lucruri comune cu TCP, ambele pentru transferul de date, dar există unele diferențe între acestea:

* TCP va garanta că datele sunt livrate, pe cand UDP nu.
* TCP este orientat spre conexiune, ceea ce înseamnă că va ține totul în legătură cu conexiunea (deschide, întreține și închide), UDP nu va face ca TCP, doar va trimite cererea, acest lucru este util în cazul Broadcast și Multicast.

În limba Java pentru protocolului UDP avem DatagramSocket, acesta fiind un tip de network socket care oferă un punct de conexiune mai mic pentru trimiterea și primirea pachetelor. Fiecare pachet trimis de la datagram socket este direcționat individual și livrat. Apoi, InetAddress object din Java reprezintă o adresă IP (Internet Protocol). Reprezentarea în octeți este pentru adresa IP. Trimiterea datelor: Scanner object, care permite controlul asupra tastaturii, iar următorul lucru este trimiterea IP-ului și a portului la server cu DatagramPacket, aceasta înseamnă că conexiunea este construită.

Logica și implementarea codului este aceleași pentru Server.

public class Client {

public static void main(String[] args) throws Exception {

DatagramSocket socket = new DatagramSocket();

InetAddress ip = InetAddress.getLocalHost();

byte sendByte[];

byte receiveByte[] = new byte[1024];

while (true) {

//Data Sending

Scanner input = new Scanner(System.in);

System.out.print("Client:");

String sendMessage = input.nextLine();

sendByte = sendMessage.getBytes();

DatagramPacket sendDatagramPacket = new DatagramPacket(sendByte, sendByte.length, ip, 8082);

socket.send(sendDatagramPacket);

//Data Receiving

DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveByte, receiveByte.length);

socket.receive(receiveDatagramPacket);

String receiveMessage = new String(receiveDatagramPacket.getData());

receiveMessage = receiveMessage.trim();

System.out.println("Server:" + receiveMessage);

}

}

}

public class Server {

public static void main(String[] args) throws Exception {

DatagramSocket serverSocket = new DatagramSocket(8082);

System.out.println("[Server started... Waiting for client to connect]");

byte receiveByte[] = new byte[1024];

byte sendByte[];

while (true) {

//data receiving

DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveByte, receiveByte.length);

serverSocket.receive(receiveDatagramPacket);

String receivedMessage = new String(receiveDatagramPacket.getData());

receivedMessage = receivedMessage.trim();

System.out.println("Client:" + receivedMessage);

//data sending

Scanner input = new Scanner(System.in);

System.out.print("Server:");

String sendMessage = input.nextLine();

sendByte = sendMessage.getBytes();

InetAddress ip = receiveDatagramPacket.getAddress();

int port = receiveDatagramPacket.getPort();

DatagramPacket sendDatagramPacket = new DatagramPacket(sendByte, sendByte.length, ip, port);

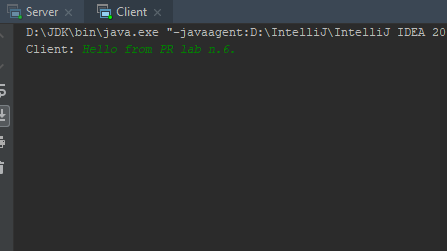
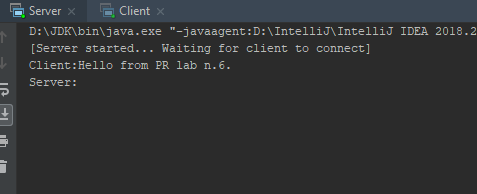
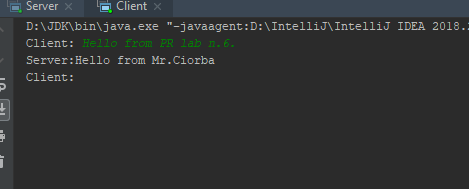
serverSocket.send(sendDatagramPacket);

}

}

}

**Output:**

[](https://user-images.githubusercontent.com/43058513/56796676-bb762a80-681b-11e9-93d6-ae8d7b7f0f97.png) [](https://user-images.githubusercontent.com/43058513/56796709-cb8e0a00-681b-11e9-905b-45c6ea842c86.png) [](https://user-images.githubusercontent.com/43058513/56796757-e2346100-681b-11e9-8492-27d1379fb42f.png)

**Concluzie:**

În această lucrare de laborator am obținut abilități de a opera cu Sockets și cu UDP-Protocol. Sockets în programare au câteva avantaje precum flexibilitate, trafic scăzut la internet și nu necesită verificări privind furnizarea datelor. Comunicarea bazată pe socket permite trimiterea numai a pachetelor raw data între aplicații. Atât partea Client, cât și Server trebuie să furnizeze mecanisme pentru a face datele utile în orice mod.

Conform documentației și experienței de lucru cu protocoale, UDP nu trimite mesajele si pachetele ca TCP. UDP este util în cazurile în care latența este critică ceea ce înseamnă că comunicarea dintre mașini nu ar trebui să aibă unele întârzieri.