Ministerul Educaţiei al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatica si Microelectronica Departamentul Ingineria Software si Automatica

**RAPORT**

Lucrarea de laborator nr.6 *Programarea in retea*

## *Tema:* Aplicație Client-Server utilizind UDP

A efectuat: st.gr. SI-171 , Buga Grigore

A verificat:

asist. univ. Cebotari Daria

Chisinau 2020

Scopul lucrarii:

Să se creeze o aplicație Client-Server UDP utilizind Sockets API

Client Server UDP Group-Chat

Am efectuat un program care implementeaza un simplu group chat, utilizind UDP

Se pot conecta pina la 3 persoane concomitent, fiecare poate vedea mesajele altor utiliziatori cu numele acestora, la fel poate expedia mesaje catre toti participantii group-chat-ului

Cod sursa:

GroupChat.java

import java.net.\*;  
import java.io.\*;  
import java.util.\*;  
public class GroupChat  
{  
 private static final String *TERMINATE* = "Exit";  
 static String *name*;  
 static volatile boolean *finished* = false;  
 public static void main(String[] args)  
 {  
 if (args.length != 2)  
 System.*out*.println("Two arguments required: <multicast-host> <port-number>");  
 else  
 {  
 try  
 {  
 InetAddress group = InetAddress.*getByName*(args[0]);  
 int port = Integer.*parseInt*(args[1]);  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter your name: ");  
 *name* = sc.nextLine();  
 MulticastSocket socket = new MulticastSocket(port);  
  
 // Since we are deploying  
 socket.setTimeToLive(0);  
 //this on localhost only (For a subnet set it as 1)  
  
 socket.joinGroup(group);  
 Thread t = new Thread(new  
 ReadThread(socket,group,port));  
  
 // Spawn a thread for reading messages  
 t.start();  
  
 // sent to the current group  
 System.*out*.println("Start typing messages...\n");  
 while(true)  
 {  
 String message;  
 message = sc.nextLine();  
 if(message.equalsIgnoreCase(GroupChat.*TERMINATE*))  
 {  
 *finished* = true;  
 socket.leaveGroup(group);  
 socket.close();  
 break;  
 }  
 message = *name* + ": " + message;  
 byte[] buffer = message.getBytes();  
 DatagramPacket datagram = new  
 DatagramPacket(buffer,buffer.length,group,port);  
 socket.send(datagram);  
 }  
 }  
 catch(SocketException se)  
 {  
 System.*out*.println("Error creating socket");  
 se.printStackTrace();  
 }  
 catch(IOException ie)  
 {  
 System.*out*.println("Error reading/writing from/to socket");  
 ie.printStackTrace();  
 }  
 }  
 }  
}  
class ReadThread implements Runnable  
{  
 private MulticastSocket socket;  
 private InetAddress group;  
 private int port;  
 private static final int *MAX\_LEN* = 1000;  
 ReadThread(MulticastSocket socket,InetAddress group,int port)  
 {  
 this.socket = socket;  
 this.group = group;  
 this.port = port;  
 }  
  
 @Override  
 public void run()  
 {  
 while(!GroupChat.*finished*)  
 {  
 byte[] buffer = new byte[ReadThread.*MAX\_LEN*];  
 DatagramPacket datagram = new  
 DatagramPacket(buffer,buffer.length,group,port);  
 String message;  
 try  
 {  
 socket.receive(datagram);  
 message = new  
 String(buffer,0,datagram.getLength(),"UTF-8");  
 if(!message.startsWith(GroupChat.*name*))  
 System.*out*.println(message);  
 }  
 catch(IOException e)  
 {  
 System.*out*.println("Socket closed!");  
 }  
 }  
 }  
}

Readthread.class

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by Fernflower decompiler)  
//  
  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.InetAddress;  
import java.net.MulticastSocket;  
  
class ReadThread implements Runnable {  
 private MulticastSocket socket;  
 private InetAddress group;  
 private int port;  
 private static final int MAX\_LEN = 1000;  
  
 ReadThread(MulticastSocket socket, InetAddress group, int port) {  
 this.socket = socket;  
 this.group = group;  
 this.port = port;  
 }  
  
 public void run() {  
 while(!GroupChat.finished) {  
 byte[] buffer = new byte[1000];  
 DatagramPacket datagram = new DatagramPacket(buffer, buffer.length, this.group, this.port);  
  
 try {  
 this.socket.receive(datagram);  
 String message = new String(buffer, 0, datagram.getLength(), "UTF-8");  
 if (!message.startsWith(GroupChat.name)) {  
 System.out.println(message);  
 }  
 } catch (IOException var5) {  
 System.out.println("Socket closed!");  
 }  
 }  
  
 }  
}

**Intrebari:**

• **Ce este un protocol orientat pe conexiune ?**

Un protocol orientat pe conexiune permite ca un flux de octeți trimiși de pe o mașină să ajungă fără erori pe orice altă mașină din inter-rețea.

**• Ce tipuri de aplicații beneficiază in general de utilizarea protocolului TCP ?**

Aplicațiile majore de internet, cum ar fi World Wide Web, e-mail, administrare la distanță și transfer de fișiere se bazează pe TCP, care face parte din stratul de transport al suitei TCP / IP. SSL / TLS se execută adesea pe TCP.

**• Cum TCP garantează că datele vor fi transmise cu succes ?**

Tehnica fundamentala, consta în faptul ca receptorul răspunde cu un mesaj de confirmare (acknowledgement) de fiecare data când primește un pachet de date. Expeditorul păstrează o copie a fiecarui pachet trimis, și așteaptă confirmarea înainte de a trimite pachetul următor. Expeditorul păstrează, de asemenea, și un timer, atunci când pachetul a fost trimis, și va relua retransmiterea pachetului în cazul în care timer-ul expira iar confirmarea recepției întârzie sa apară. Contorul de timp este necesar în cazul în care un pachet se pierde sau este deteriorat.

**• Diferența dintre blocking si non-blocking sockets**

Socket-urile pot fi în modul de blocare sau de blocare, care este controlat prin apelarea funcției ioctlsocket. În modul de blocare, apelurile API socket, recv, send, connect (doar TCP) și accept (doar TCP) vor bloca la nesfârșit până când acțiunea solicitată a fost efectuată. În modul care nu blochează, aceste funcții revin imediat. select se va bloca până când priza este gata. selectează un parametru de timp care controlează timpul de așteptare pentru finalizarea acțiunii sau o eroare returnată.

**• Diferența dintre blocking multithreaded și non-blocking single thread socket**

**• Cum are loc procesul TCP Three Way Handshake ?**

Pasul 1 (SYN): În primul pas, clientul dorește să stabilească o conexiune cu serverul, astfel încât trimite un segment cu SYN (Sincronizați numărul de secvență) care informează serverul că clientul este probabil să înceapă comunicarea și cu ce număr de secvență începe segmentele. cu

Pasul 2 (SYN + ACK): Serverul răspunde la solicitarea clientului cu biți de semnal SYN-ACK setat. Recunoaștere (ACK) semnifică răspunsul segmentului pe care l-a primit, iar SYN semnifică cu ce număr de secvență este posibil să înceapă segmentele cu

Pasul 3 (ACK): în partea finală clientul recunoaște răspunsul serverului și amândoi stabilesc o conexiune fiabilă cu care va începe transferul de date efectiv

**• Numiti cele 4 apeluri de sistem necesare pentru a crea un server TCP**

**• Care este rolul metodei bind() ?**

bind () asociază o priză cu o adresă. Când o socket este creată cu socket (), este dată doar o familie de protocoale, dar nu i se atribuie o adresă. Această asociere trebuie efectuată înainte ca socket-ul să poată accepta conexiuni de la alte gazde.

**• Care este rolul metodei accept() ?**

accept () returnează noul descriptor socket pentru conexiunea acceptată sau valoarea -1 dacă apare o eroare. Toate comunicările ulterioare cu gazda de la distanță au loc acum prin intermediul acestui nou soclu.

**• Care este adresa de loopback IPv4 și care este rolul ei ?**

Loopback este un canal de comunicare cu un singur sfarsit. Retelele TCP/IP prezintă un loopback care permite software-ului client sa comunice cu software-ul server pe același computer. Utilizatorii pot specifica o adresa IP, de obicei 127.0.0.1, care va indica înapoi la configurația rețelei TCP/IP a computerului. Intervalul de adrese pentru funcționalitatea loopback este intervalul de la 127.0.0.0 pana la 127.255.255.255. Similar cu ping, loopback permite utilizatorului sa-si testeze propria rețea pentru a se asigura ca stack-ul IP funcționează corect.

**• Puteți imbunătăți performanța aplicației prin dezactivarea algoritmului Nagle ?**

Algoritmul lui Nagle este un mijloc de îmbunătățire a eficienței rețelelor TCP / IP prin reducerea numărului de pachete care trebuie trimise prin rețea.

**• Ce instrumente listează socket-urile TCP deschise in sistemele de operare Windows și Linux ?**

Netstat, arp, ping, route