Universitatea Tehnica a Moldovei
Facultatea Calculatoare Informatică și Microelectronică

Departamentul ISA

Raport

Lucrarea de laborator Nr. 6
la Programarea în rețea

Tema: Proiectare și progrmare aplicație client server

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

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 între componente.

Link repozitoriu distant GIT: https://github.com/GorduzDaniel/Lab6_PR

Noțiuni teoretice

Serverul este o aplicatie 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 ofera. Deci daca 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]

Mersul lucrării

Pentru început în partea server, se creează un obiect de tip ServerSocket, în constructorul căruia îi specificăm portul pe care va lucra serverul propriu-zis. Într-un ciclu infinit se include ca obiectul ServerSocket să accepte conexiuni. În cazul când a avut loc o conexiune, se va porni un fir de execuție nou, pentru partea client conectată. În figura 1 este prezentată clasa Server.

```
public class Server {
    private final static int PORT = 4998;

    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket socket = null;

        try {
            serverSocket = new ServerSocket(PORT);
            System.out.println("Socket created!");
        } catch (IOException e) {
            System.out.println("Socket not created. Shutting down!");
            e.printStackTrace();
        }

        System.out.println("Server is listening...");

        while (true){
            try {
                if (serverSocket != null) {
                    socket = serverSocket.accept();
                }
                System.out.println("Connection accepted!");
                new TServer(socket).start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figura 1 – Clasa server

În figura 2 este prezentat fiecare fir de execuție pentru partea client. Astfel în constructor se primește obiectul de tip Socket, care va servi pentru întreschimbarea de date între client și server. Acest lucru este are loc cu ajutorul fluxurilor între clase cu ajutorul obiectelor *BufferedReader* și *PrintWriter*.

Clasa va moșteni funcționalul clasei *Thread*. În metoda moștenită *run()*, este implimentat funcționalul pentru pentru fiecare fir de execuție, care va trimite răspunsuri la partea client. La fel ca și în cazul anterior se creează un ciclu infinit, în care mereu se va citi ceea ce trimite partea client. Punctul de ieșire din acest ciclu este momentul în care clientul apelează la comanda *quit.* În acel moment socket-ul se va închide, și respectiv comunicarea va fi terminată.

```
public class TServer extends Thread {
    private Socket socket = null;
    String username = null;
    String password = null;

    public TServer(Socket socket) { this.socket = socket; }


    public void run() {
        InputStream inp;
        BufferedReader brinp;
        PrintWriter out;
        FileWriter fileWriter;
        boolean isLoggedIn = false;
        boolean isCounted = false;
        boolean isBought = false;
        int counter;
        try {
            inp = socket.getInputStream();
            brinp = new BufferedReader(new InputStreamReader(inp));
            out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
        } catch (IOException e) {
            return;
        }
        String line;
        outer:
        while (true) {
            try {
                line = brinp.readLine();
```

Figura 2 – Fir de execuție care răspunde părții client

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*.

```java
public class Client {
    public static void main(String[] args) throws IOException {
        InetAddress address = InetAddress.getLocalHost();
        Socket socket;
        String line;
        BufferedReader bufferedReader;
        BufferedReader in;
        PrintWriter out;
        boolean isLoggedIn = false;
        socket = new Socket(address,  port: 4998);
        bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(),  autoFlush: true);

        System.out.println("Client address is: " + address);
        System.out.println("Connection established! Write QUIT to end.");
        System.out.println("To see the list of available commands, just type: commands");
        String response;


        line = bufferedReader.readLine();
        while (!line.equalsIgnoreCase( anotherString: "QUIT")){
            if(line.equals("commands")){
                out.println(line);
                for (int i = 0; i < 8; i++) {
                    response = in.readLine();
                    System.out.println("SERVER said: " + response);
                }
                line = bufferedReader.readLine();
```

Figura 3 – Implimentarea părții Client

Pentru ușurarea interacțiunii cu utilizatorii a mai fost creată o interfață grafică, pe lângă cea de consolă, care este prezentată în figura 4.

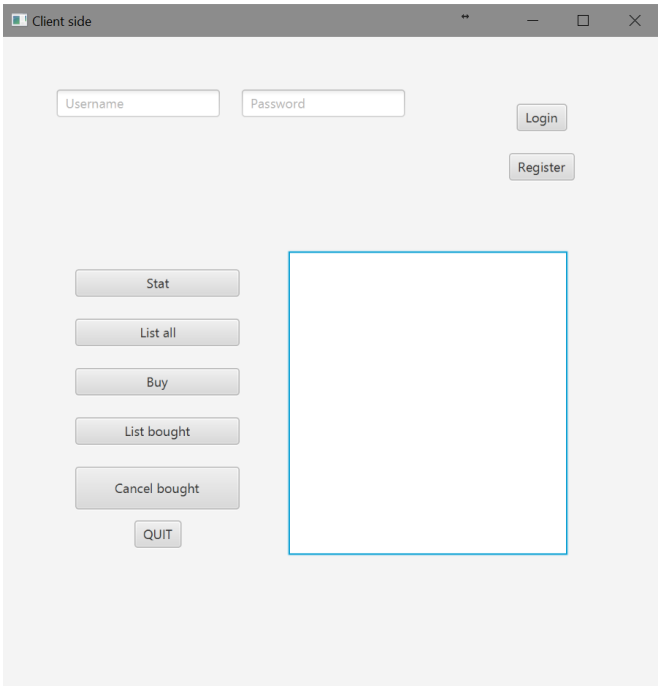

Figura 4 – Interfață grafică pentru partea client

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 cu ajutorul firelor de execuție, 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.

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