# STUDENTSKA PLATFORMA ZA TESTIRANJE DISTRIBUIRANIH APLIKACIJA

PROJEKTNI ZADATAK ZA PREDMET: DISTRIBUIRANI PROCESI

Leonora Gašpar

Silva Haberl

## UVOD

Zadatak je bio testirati klase koje omogućuju transparentno povezivanje i komuniciranje procesa u praktikumu. Osim toga, trebalo je nadograditi izmjeniti postojeće klase kako bi one omogućavale drugim studentima da svoje distribuirane programe testiraju u realističnim uvijetima, na više računala. U ovom seminaru opisujemo tehnologije vezane za rješenje, te dio programskog rješenja.

## RJEŠENJE

**LAN (Local Area Network)** namijenjena je povezivanju računala i drugih mrežnih uređaja na manjim udaljenostima, npr. u okviru jednog ureda, zgrade, postrojenja ili kuće. Komunikacija se odvija preko TCP/IP protokola. LAN omogućava dijeljenje podataka, uređaja kao i programa.

U računarskom praktikumu računala su povezana u LAN mrežu.

Za povezivanje i pokretanje procesa na više računala, bilo je potrebno sljedeće: Odrediti računalo i njegov hostname koje će glumiti server , te port na kojem će osluškivati klijentska računala, koja se žele spojiti na Socket.

Sve što je za to trebalo promijeniti u Java kodu s predavanja jest u Symbols.java na svakom računalu sa kojim radimo, pod nameServer stavimo hostname računala koje predstavlja server, te serverPort postaviti na neki cijeli broj veći od 1024 jer su svi manji zauzeti za sistemske zadatke. U klasi NameServer uz male promjene ispisa postojećih procesa koji su zapisani u NameTable, možemo pokrenuti NameServer.java.

```
public static void main(String[] args) {
        NameServer ns = new NameServer();
        System.out.println("NameServer started");
                System.out.println("----------------------------------------");
        try {
            ServerSocket listener = new ServerSocket(Symbols.ServerPort);
            while (true) {
                Socket aClient = listener.accept();
                ns.handleclient(aClient);
                                System.out.println(" Name table elements:");
                for (int i = 0; i < 10; i++) {
                    String name = ns.table.getName(i);
                    String hostName = ns.table.getHostName(i);
                    int port = ns.table.getPort(i);
                    if(name != null)
                       System.out.println("   name: " + name + ", hostName: "
                                                    + ", port: " + port );
                }
                aClient.close();
            }
        } catch (IOException e) {
            System.err.println("Server aborted:" + e);
        }
    }
```

```
}
```

Za lijepši ispis LinkerTestera , u klasi Name ispisujemo svaki puta što smo dodali u zablicu svih procesa:

```
public class Name {
        BufferedReader din;
        PrintStream pout;
    public void getSocket() throws IOException {
        Socket server = new Socket(Symbols.nameServer, Symbols.ServerPort);
        din= new BufferedReader(new InputStreamReader(server.getInputStream()));
        pout = new PrintStream(server.getOutputStream());
    }
    public int insertName(String name, String hname, int portnum)
            throws IOException {
        getSocket();
        System.out.println("Insert name: " + name + ", host name: " + hname +
                                        ", port number: " + portnum);
        pout.println("insert " + name + " " + hname + " " + portnum);
        pout.flush();
        return Integer.parseInt(din.readLine());
    }
    public PortAddr searchName(String name) throws IOException {
        getSocket();
        pout.println("search " + name);
        pout.flush();
        String result = din.readLine();
        StringTokenizer st = new StringTokenizer(result);
        int portnum = Integer.parseInt(st.nextToken());
        String hname = st.nextToken();
        return new PortAddr(hname, portnum);
    }
```

Pregledniji ispis poruka u klasi LinkerTester dobijemo:

```
public class LinkerTester {
public static void main(String[] args) {
        Linker comm = null;
        Msg m;
        try {
            String baseName = args[0];
            int myId = Integer.parseInt(args[1]);
            int numProc = Integer.parseInt(args[2]);
                      System.out.println("LinkerTester started:");
                      System.out.println(" base name: " + baseName + ", id: " +
                                  myId + ", number of processes: " + numProc);
                      System.out.println("---------------------------------");
            comm = new Linker(baseName, myId, numProc);
            for (int i = 0; i < numProc; i++)
                if (i != myId){
                    comm.sendMsg(i, "my_tag", "Uspješno ste se povezali procese
                                    na vise racunala");
                    System.out.println(" >  Sent message: ");
                    System.out.println(" >> To: " + i + ", from: " + myId);
                    System.out.println(" >> Tag: " + "my_tag");
                    System.out.println(" >> Content: " + "Uspješno ste se
                                    povezali procese na vise racunala");
                }
            for (int i = 0; i < numProc; i++)
                if (i != myId)
                    m = comm.receiveMsg(i);
        }
  catch (Exception e) {  System.err.println(e);  }
```

```
    }
}
```

**SSH (engl.Secure Shell)** je mrežni protokol koji korisnicima omogućuje uspostavu sigurnog komunikacijskog kanala između dva računala putem nesigurne računalne mreže. SSH obično koristi port 22 tcp/udp, što je na Unix računalima definirano u datoteci /etc/services. SSH se obično koristi za prijavu s jednog računala na drugo i izvršavanje naredbi na drugom računalu.

Kako bismo pokrenuli na svakom od klijentskih računala istovremeno procese sa onog koji predstavlja server, potreban nam je ssh. OpenSSH Server omogućuje udaljeni pristup sustavu koristeći protokol SSH. Iz komandne ljuske on se može instalirati putem sljedeće naredbe (koja instalira pakete openssh-server i openssh-client):

**$ sudo apt-get install ssh**

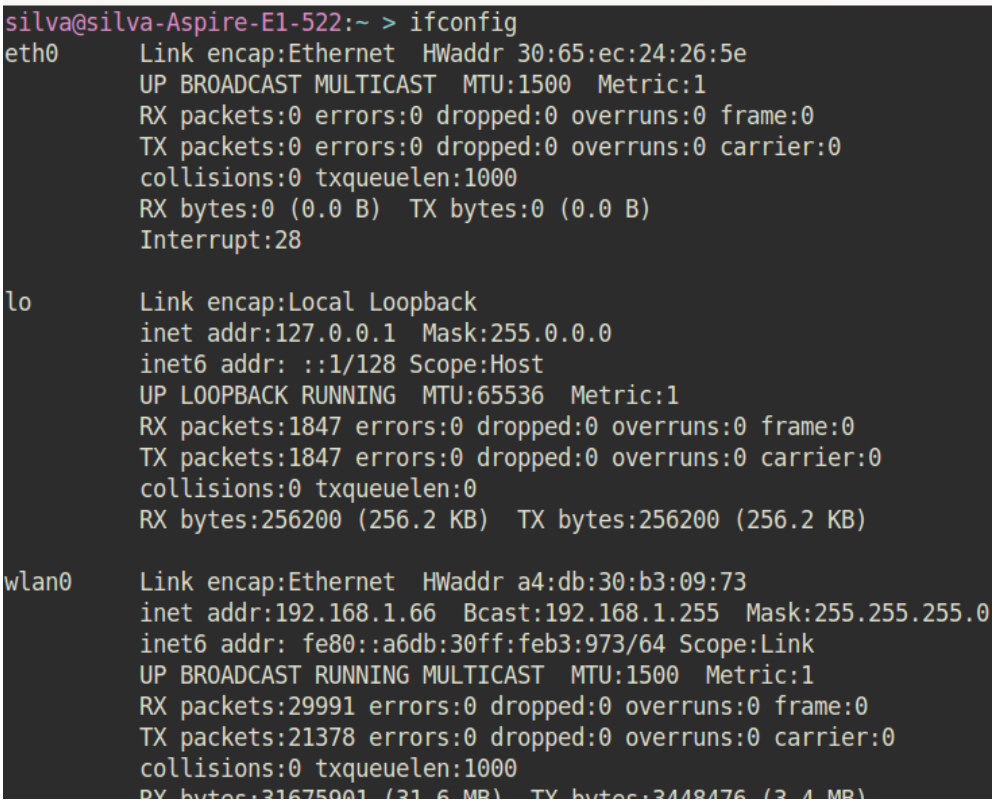
Da bi se uspostavila veza na SSH-server, potrebno je koristiti neki SSH-klijent. Na Linuxu je to OpenSSH Client koji omogćuje rad u tekstualnom sučelju (komandnoj ljuski). Primjer spajanja na računalo 'host' za korisnika 'ivo' sa ssh:

**$ ssh** ivo@host

Želimo li vidjeti IP adresu našeg lokalnog računala, odnosno sva mrežna sučelja preko kojih se može komunicirati sa trenutnim računalnom na Linuxu koristimo naredbu:

**$ ifconfig.**

Dobivamo ispis kao na sljedećoj slici:




U računarskom praktikumu računala su povezana međusobno preko eth0 (Ethernet), gdje bi IP pojedinog računala pročitali pod inet 192.168.....

Da su računala povezana preko Wi-Fi (WirelessLAN) čitali bi pod wlan0 inet 192.168....

## SKRIPTE

Standaran primjer skriptog jezika za specifično okruženje: **Bash** za Unix operativne sustave. Naredba za otkrivanje gdje se bash interpreter nalazi :

**$ which bash**

postavljanje skripte da bude izvršna datoteka

**$ chmod +x hello_world.sh**

Izvršavanje bash skripte

**./hello_world.sh** ili **source hello_world.sh**

Za naš program koristimo 3 jednostavne skripte.

### Prva skripta:

```
#!/bin/bash
PATH=/opt/jdk1.8.0_65/bin:$PATH
export PATH
javac *.java
java NameServer
```

Prva linija svake bash skripte započinje znakom *#!* "she-bang" ili "hash-bang". Nakon toga slijedi /bin/bash put kojim pokazujemo Unix sistemu kojim interpreterom (programom) želimo pokrenuti datoteku. Zapravo će se pri svakom pokretanju skripte, pozivati */bin/bash naziv_skripte.sh*.

Sljedeće dvije linije omogućuju kompiliranje i pokretanje java datoteka bez navođenja puta do direktorija gdje je java na računalu instalirana.

Riječ 'export' omogućuje da pri svakom pokretanju ove skripte, u varijablu PATH dodan je put do jave, nastavljajući se na stari PATH. Kada to ne bi dodali u PATH, kompilirali i pokretali bi java programe sa: */opt/jdk1.8.0_65/bin javac ...* odnosno */opt/jdk1.8.0_65/bin java ... (*/opt/jdk1.8.0_65/bin je put do jave u računarskom praktikumu*).*

javac *.java kopilira sve java datoteke u trenutnom direktoriju, gdje se sktipta nalazi.

Posljednja linija skripte pokrece NameServer program (datoteku).

Rezultat pokretanja prve skripte: Server čeka na procese sa klijentskih računala da zatraže komunikaciju.

## Druga skripta:

```
#!/bin/bash
PATH=/opt/jdk1.8.0_65/bin:$PATH
export PATH
mate-terminal -e 'bash bashC.sh 0 2 user@host1'
mate-terminal -e 'bash bashC.sh 1 2 user@host2'
```

Druga skripta za razliku od prve ima svrhu otvaranja novih terminala (komandnih ljuski) u kojima izvršavamo treću skriptu bashC.sh proslijeđujući joj dva parametra.

Naredba *mate-terminal* otvara novi terminal (komandnu ljusku). Oznaka '-e' znači izvršavanje onoga što slijedi u stringu koji slijedi iza oznake.

U stringu se nalazi naredba za pokretanje skripte bashC.sh sa parametrima : broj procesa trenutnog klijentskog računala i ukupan broj procesa koje želimo pokrenuti. Treći parametar koji navodimo je naziv klijentskog računala, tj. njegov hostname.

Otvorimo onoliko terminala (pokrenemo skripti) koliko želimo povezanih procesa.

## Treća skripta:

```
#!/bin/bash
PATH=/opt/jdk1.8.0_65/bin:$PATH
export PATH
i=$1
N=$2
h=$3
ssh $h << EOF
   cd /.home/student1/user/public_html/dprojekt
   java LinkerTester base $i $N
EOF
$SHELL
```

U trećoj skripti 4., 5. i 6. linija uzimaju vrijednosti iz prvog, drugog i trećeg argumenta iz komandne linije i spremaju ih u varijable *i, N i h.* Argument *i* je indeks procesa koji se pokreće na klijentskom računalu čiji je hostname treći argument *h*, drugi argument *N* je ukupan broj procesa koje pokrećemo i povezujemo.

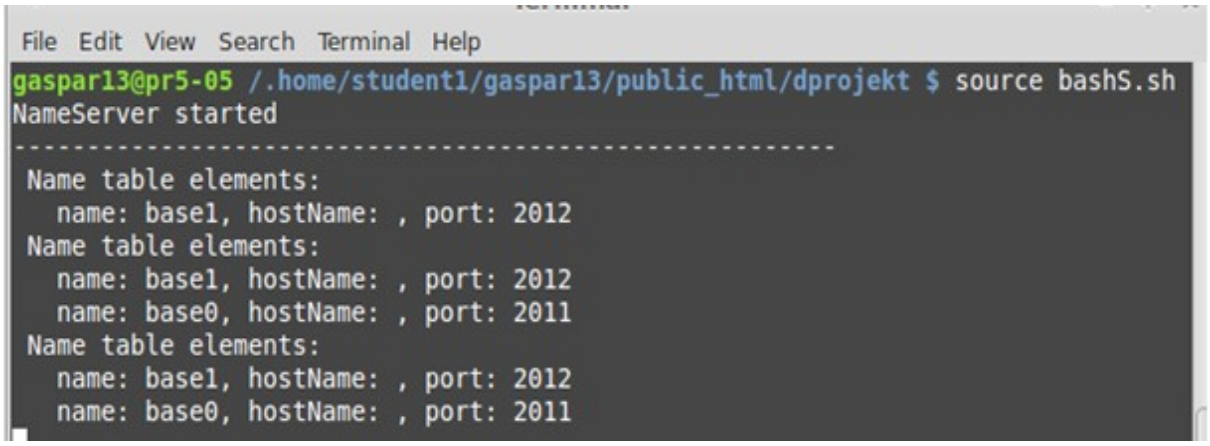
Naredbom *ssh $h* povezujemo se na računalo čiji je hostname h. Povezali smo se (ssh login) koristeći šifru (password) usera na čiji račun ulazimo. Mogli smo ukloniti unošenje passworda koristeći ključ (private-public key) koji smo generirali.

Riječ *EOF* (tag) označava početak i kraj višeretčanog stringa. Može biti bilo koja druga riječ, pod uvijetom da je ista na početku i kraju stringa. Uobičajeno se koriste velika tiskana slova. Na taj način izvršit će se naredbe ulaska u direktorij (cd /..) gdje se nalazi datoteka LinkerTester.java i pokretanje LinkerTester.java datoteke (java ...) sa zadanim argumentima.

Varijabla *$SHELL* na kraju skripte omogućuje da se nakon izvršavanja skripte terminal ne zatvori. $SHELL sadrži put (path) do interpretera koji ovdje koristimo.

Ako smo pokrenuli sve tri skripte, dobivamo sljedeći rezultat:

1.Pokrenuli smo NameServer.java



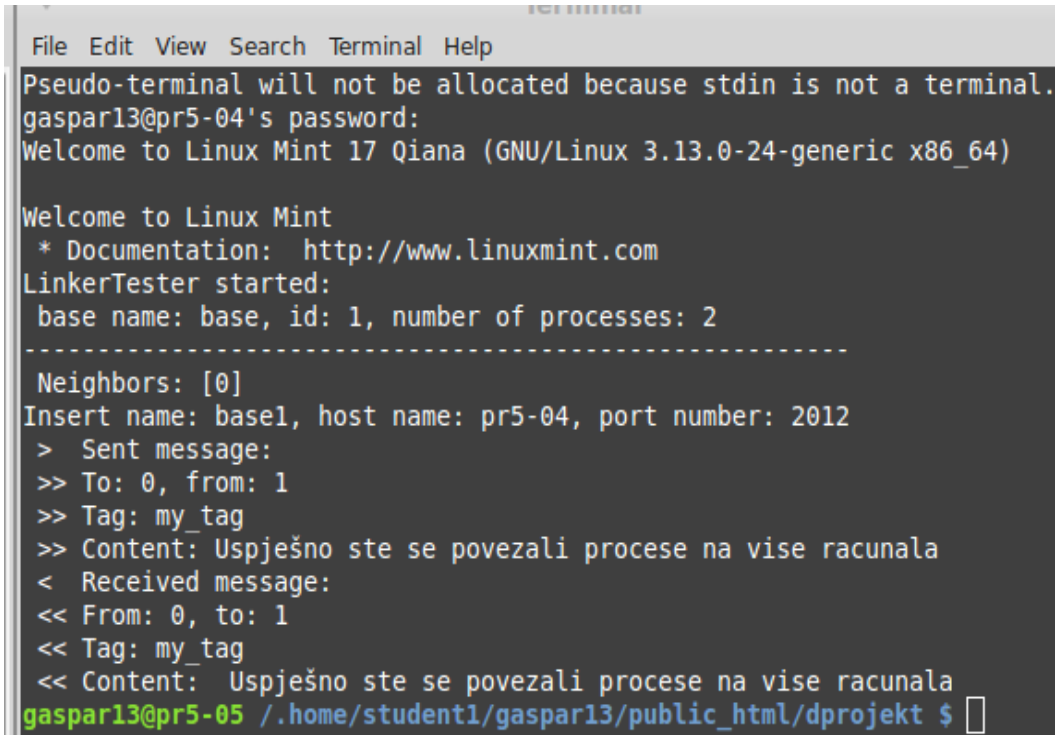

NameServer čeka na klijente da preko socketa zatraže povezivanje i slanje poruka.

NameServer će svaki proces sa njegovim podacima (ime, hostname, port) spremiti u tablicu svih procesa, NameTable. Nakon svakog poziva handleClient u klasi NameServer ispisujemo sve elemente od NameTable.

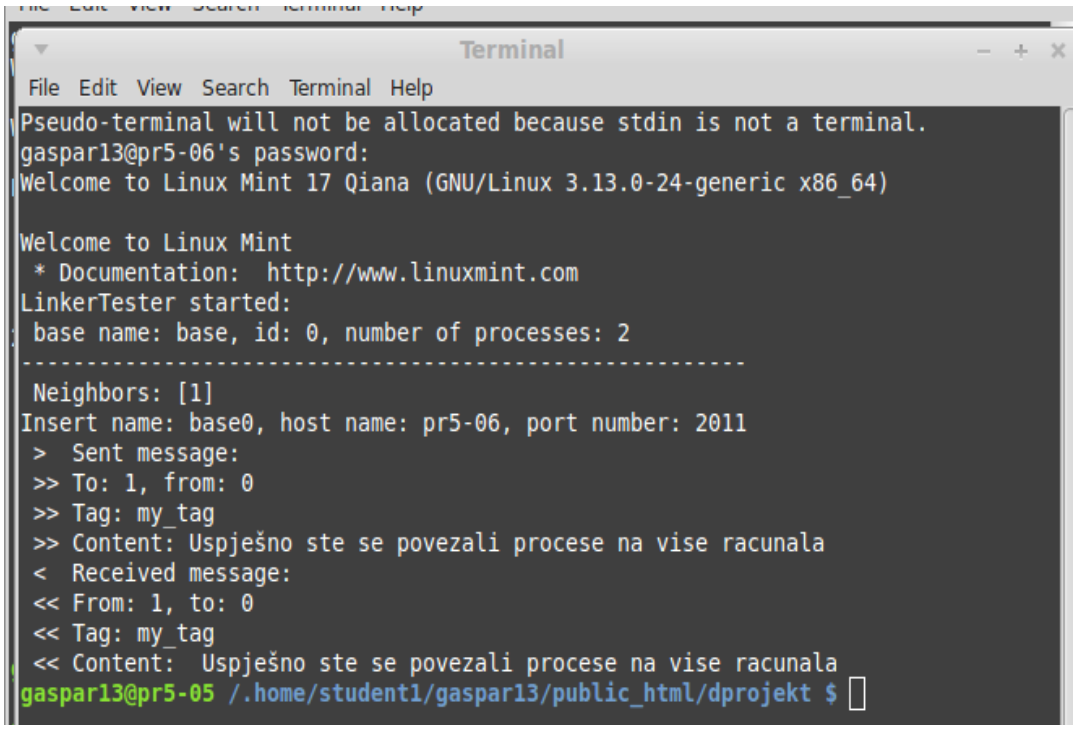
Kao što vidimo na slici, NameServer se neprekidno izvršava i čeka nove zahtjeve. S druge strane, LinkerTester se nakon izvršavanja ugasi.

Kada bismo pokrenuli LinkerTester prije pokretanja NameServer-a, dobili bi poruku o grešci.

2.Pokrenuli smo proces na klijentu pod rednim brojem 0 i 1:




Poruke su razmijenjene (Sent i Recived message):




Najprije možemo vidjeti kontrolni ispis argumenata s kojima smo pokrenuli klase LinkerTester. Zatim možemo vidjeti ispis iz Topologije.
Prilikom umetanja podataka u NameTable, ispišemo i te parametre.

Na kraju možemo vidjeti razmjenu poruka.

Time smo demonstrirali kako se klase ponašaju za dva procesa na dva različita računala.
Kada bi, na primjer, htjeli imati 10 različitih procesa na 10 računala, u drugoj skripti bismo promjenili :

```
#!/bin/bash
PATH=/opt/jdk1.8.0_65/bin:$PATH
export PATH

mate-terminal -e 'bash bashC.sh 0 10 user@host1'
mate-terminal -e 'bash bashC.sh 1 10 user@host2'
.       .       .       .       .       .       .       .       .
.       .       .       .       .       .       .       .       .
.       .       .       .       .       .       .       .       .
mate-terminal -e 'bash bashC.sh 9 10  user@host10'
```

**Primjer:** Chat (7.poglavlje + prošlogodišnja GUI aplikacija chat-a)

Naše prilagođene klase primjenili smo na prošlogodišnjem projektu za Chat koji koristi klase iz 7.poglavlja. Zamjenili smo odgovarajuće klase iz tog rješenja sa našima i uvjerili se da program zaista radi.

Pokretanje je ekvivalentno prethodnom primjeru,  samo što umjesto naredbe za pokretanje LinkerTester-a, pokreće se :

***java Chat $i $N causal*** (*kauzalni uređaj)*

Dakle, dobivena platforma omogućuje drugim programima vezanim za ovaj kolegij, izvođenje u realističnim uvjetima.

Literatura:

Garg V.K. Concurrent and Distributed Computing in Java. Wiley – IEEE Press, New Yersey, 2004