Klijentski socket-i

Mrežno i distribuirano programiranje

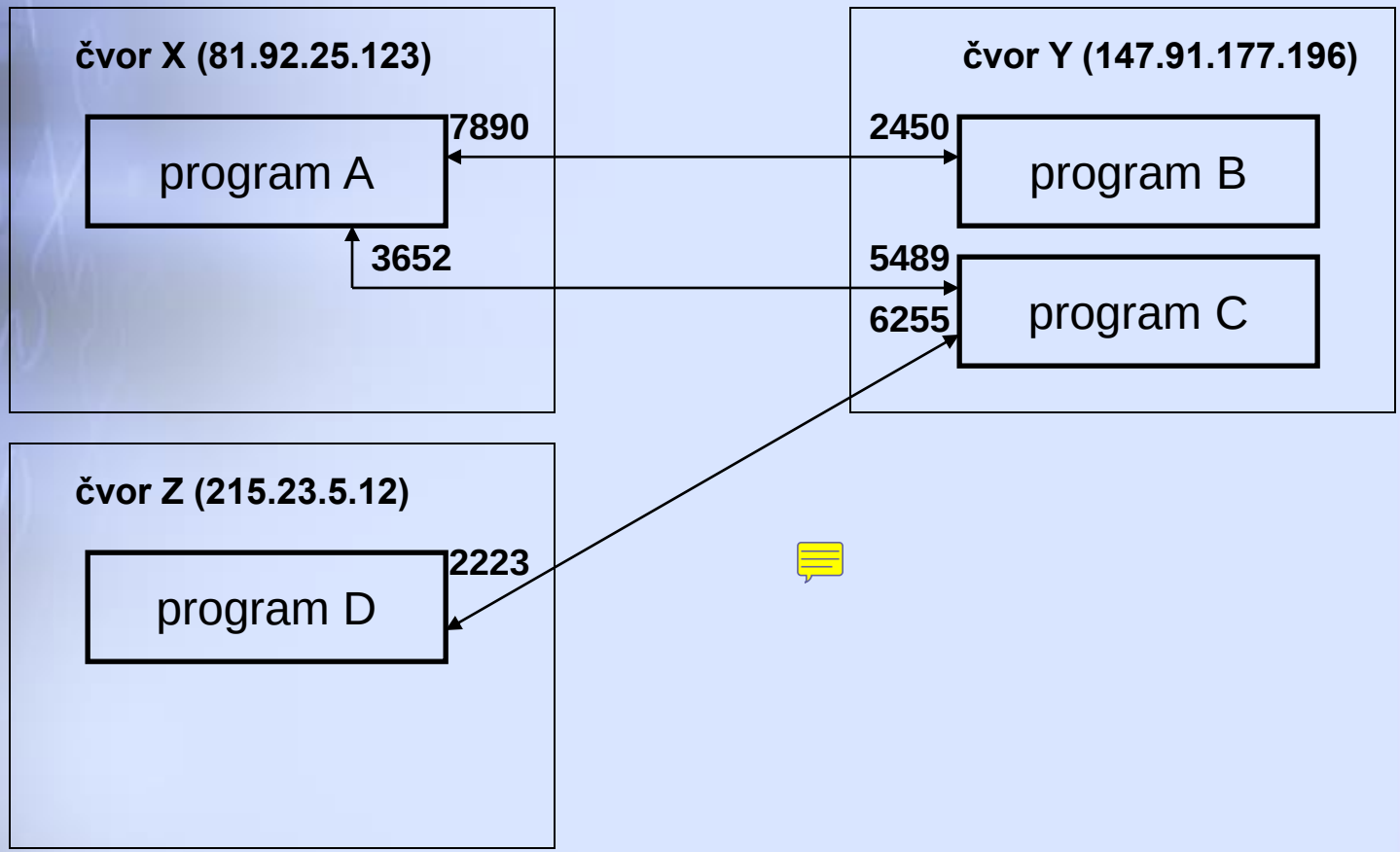
Socket

- pojam *socket-a*: uređeni par (IP adresa, port) jednog učesnika u komunikaciji
- veza između dvije aplikacije koje komuniciraju preko mreže uspostavlja se preko dva *socket*-a, pri čemu se na strani svake aplikacije nalazi po jedan *socket*

- *socket*-i omogućavaju da se mrežna konekcija tretira kao tok u koji podaci mogu biti upisani i iz kojeg podaci mogu biti pročitani.

Pojam Socket-a

- veza između 2 programa, ne 2 računara



Klasa Socket

- klasa Socket predstavlja apstrakciju krajnje tačke u komunikaciji između dvije aplikacije
- Java klijentske aplikacije obično koriste klasu Socket na sljedeći način: kreiraju objekat klase Socket u cilju povezivanja na udaljenu serversku aplikaciju, a nakon toga koriste kreirani Socket objekat za dobijanje referenci na ulazni i izlazni tok
- ekvivalentan proces obavlja se i na strani servera
- klijent i server mogu slati podatke jedan drugom - *full-duplex* konekcija, što znači da obe strane u komunikaciji mogu slati i primati podatke istovremeno
- o kakvim podacima se radi zavisi od protokola kojim klijent i server komuniciraju
  - tekstualni protokoli
    - mogu prenositi podatke u različitim formatima, poput XML (*eXtensible Markup Language*) i JSON (*JavaScript Object Notation*) formata.
  - binarni

Klasa Socket

- Konstruktori:
  - jedan bez argumenata
  - ostali s argumentima

```
Socket()                                                          // 1
Socket(InetAddress address, int port)                             // 2
Socket(InetAddress address, int port, InetAddress localAddr,
int localPort)                                                    // 3
Socket(String host, int port)                                     // 4
Socket(String host, int port, InetAddress localAddr, int
localPort)                                                        // 5
Socket(Proxy proxy)                                               // 6
```

Klasa Socket

- važne metode:
  - InputStream getInputStream()
  - OutputStream getOutputStream()
  - void connect(SocketAddress endpoint)
  - void connect(SocketAddress endpoint, int timeout)
  - void close()
  - ...

Uspostavljanje konekcije

- korištenjem konstruktora

```
public class PortScanner {
    private static int minPort = 1;
    private static int maxPort = 443;
    private static String host = "smtp.elta-kabel.com";

    public static void main(String[] args) {
        for(int i=minPort; i<= maxPort; i++) {
            try {
                Socket s = new Socket(host, i);
                System.out.println("Port " + i + " open");
                s.close();
            } catch (UnknownHostException e) {
                System.err.println("Unknown host: " + host);
                break;
            } catch (IOException e) {
//              System.err.println("Port" + i + " not opened.");
            }
        }
    }
}
```

Čitanje iz mrežne konekcije

- metoda getInputStream()
- Primjer: daytime protokol

```
public class DayTimeClient {
    private static String dtServer = "time.nist.gov";
    private static int PORT = 13;

    public static void main(String[] args) {
        try(Socket client = new Socket(dtServer, PORT)){
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    client.getInputStream(), "ASCII"));
            br.readLine();
            String dateTime = br.readLine();
            System.out.println(dateTime);
            br.close();
        } catch (UnknownHostException e) {
            System.err.println("Host Unknown");
        } catch (IOException e) {
            System.err.println("IO Exception occured");
        }
    }
}
```

Upis u mrežnu konekciju

- metoda getOutputStream()
- Primjer: SMTP protokol

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        client.getInputStream()));
PrintWriter out =
    new PrintWriter(
        new BufferedWriter(
            new OutputStreamWriter(
                client.getOutputStream())), true);
String welcomeMsg = in.readLine();
System.out.println("S: " + welcomeMsg);
out.println("QUIT");
System.out.println("C: QUIT");
String byeMsg = in.readLine();
System.out.println("S: " + byeMsg);
in.close();
out.close();
```

Podešavanje opcija *socket*-a

- Konstruktor Socket klase bez argumenata

```
public class DayTimeClientNoArgSocket {
    private static String dtServer = "time.nist.gov";
    private static int PORT = 13;

    public static void main(String[] args) {
        try(Socket client = new Socket()){
            SocketAddress sa = new InetSocketAddress(dtServer, PORT);
            client.connect(sa);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    client.getInputStream(), "ASCII"));
            br.readLine();
            String dateTime = br.readLine();
            System.out.println(dateTime);
            br.close();
        } catch (UnknownHostException e) {
            System.err.println("Host Unknown");
        } catch (IOException e) {
            System.err.println("IO Exception occured");
        }
    }
}
```

- preklopljena metoda connect>
  - void connect(SocketAddress endpoint)
  - void connect(SocketAddress endpoint, int timeout)

Podešavanje opcija *socket*-a

- TCP_NODELAY opcijom omogućava se slanje paketa što je brže moguće, bez obzira na njihovu veličinu – isključuje se baferovanje
- SO_LINGER opcijom specificira se šta se dešava sa paketima koji još uvijek nisu poslati u trenutku kada se *socket* zatvara – ako *linger* vrijeme ima pozitivnu vrijednost, onda će close metoda biti blokirana specificirani vremenski period
- SO_TIMEOUT opcijom specificira se vremenski period čekanja u slučaju poziva blokirajućih operacija
- SO_SNDBUF i SO_RCVBUF opcijama specificira se preporučena veličina bafera koji platforma koristi, pri čemu se SO_SNDBUF opcijom specificira preporučena veličina bafera koji platforma koristi za slanje podataka putem datog *socket*-a, a SO_RCVBUF opcijom specificira se preporučena veličina bafera koji platforma koristi za primanje podataka putem datog *socket*-a

Podešavanje opcija *socket*-a

- SO_KEEPALIVE opcijom specificira se slanje *keepalive* paketa u slučaju kada putem *socket*-a podaci nisu razmijenjeni određeni vremenski period
- SO_OOBINLINE opcijom specificira se da hitni podaci primljeni putem *socket*-a budu pročitani putem ulaznog toka vezanog za dati *socket*

```
client.setReuseAddress(true);
client.setKeepAlive(true);
client.setTcpNoDelay(true);
```

- SO_REUSEADDR opcijom specificira se mogućnost ponovne upotrebe adrese i porta u slučaju kada je konekcija u TIME_WAIT stanju, a nakon što je konekcija prethodno zatvorena
- IP_TOS opcijom specificira se klasa saobraćaja (tj. tip servisa) u zaglavlju IP paketa koji se šalju putem tekućeg Socket objekta

Komunikacija putem *proxy* servera

- upotreba *proxy* servera se podešava isto kao kod URL klase
- konstruktor: Socket (Proxy proxy)

```
SocketAddress pa = new InetSocketAddress("128.199.246.226", 8080);
Proxy proxy = new Proxy(Proxy.Type.HTTP, pa);
try(Socket client = new Socket(proxy)){
    SocketAddress sa = new InetSocketAddress(dtServer, PORT);
    client.connect(sa);
```

Dobijanje različitih informacija o mrežnoj konekciji

- IP adresa i broj porta udaljenog *socket*-a na koji je vezan *socket* predstavljen tekućim objektom klase Socket
- informacija o tome da li postoji ili je postojala konekcija prema udaljenom hostu
- ...

```
System.out.println(client.getRemoteSocketAddress());
System.out.println(client.getInetAddress());
System.out.println(client.getPort());
System.out.println(client.getLocalAddress());
System.out.println(client.getLocalPort());
System.out.println(client.isBound());
System.out.println(client.isConnected());
System.out.println(client.isClosed());
boolean connected = client.isConnected() && ! client.isClosed();
```