4. COMUNICAȚII JAVA FOLOSIND SOCKET

4.1. Prezentarea generală a pachetului java.net

4.1.1. Principalele clase

Acest pachet reprezintă o infrastructură puternică şi flexibilă de programare în rețea. Multe dintre clasele pachetului `java.net` fac parte din această infrastructură şi nu sunt folosite direct în programele de aplicații. Din această cauză prezentăm numai partea din structura pachetului care este direct utilizabilă în programe. Întreg pachetul este structurat pe un singur nivel; toate clasele descind direct din clasa `Object`, aşa cum se vede în fig. 4.1.

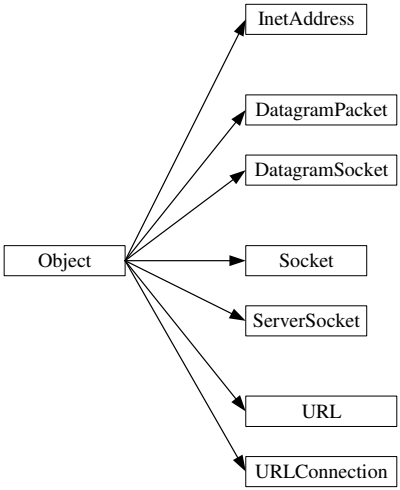

Figura 4.1. Structura pachetului `java.net`

Clasele pachetului sunt destinate să realizeze patru funcțiuni principale:

- gestiunea adreselor Internet;
- comunicarea prin UDP;
- comunicarea prin TCP;

- accesul la resursele Internet prin URL.

Tabelul următor indică analogii intuitive ale semnificațiilor acestor clase:

`InetAddress`	Purtător de adresă Internet şi IP
`DatagramPacket`	Pachet UDP - "plic purtător al unei scrisori"
`DatagramSocket`	Capăt UDP - "căsuță poştală"
`Socket`	Capăt TCP - "telefon care face apel"
`ServerSocket`	Capăt TCP - "telefon care aşteaptă apel"
`URL`	Purtăror de adresă specificată prin URL
`URLConnection`	O conexiune activă la un obiect Internet: CGI, servlet, serviciu de rețea etc.

4.1.2. Câteva servicii standard de comunicații

Comunicațiile prin socket, indiferent dacă sunt în Java sau în C/C++ sau alt suport de dezvoltare, se fac după principiul client - server. Cei doi protagonişti care comunică pot fi elaborați în limbaje diferite, singura grijă a proiectantului este să țină cont de reprezentările datelor. Mai mult, într-o astfel de comunicație este posibil ca unul dintre protagonişti să fie un serviciu standard, acesta fiind, de cele mai multe ori, partea de server.

Proiectantul unei aplicații distribuite poate să comunice cu astfel de servicii, cu condiția ca serviciul dorit să fie activ pe maşina țintă. Specificațiile (protocoalele) de comunicație pentru serviciile standard sunt publice şi sunt disponibile pe Internet sub forma unor **RFC**-uri (***R**equest **F**or **C**omments*). De asemenea, serviciile standard au asociate porturi standard de comunicare, precum şi tipurile de socket (TCP, UDP) prin care comunică. Tabelul următor prezintă câteva astfel de servicii:

Serviciul distribuit	**Port asociat**	**Socket**	**RFC de descriere**
DayTime	13	TCP UDP	867
Time	37	TCP UDP	868
Echo	7	TCP UDP	862
SMTP (***S**imple **M**ail **T**ransfer **P**rotocol*) server e-mail	25	TCP	821 (SMTP), 822 (Mail Format), 1521 (MIME), 1869 (Extended SMTP)
SSMTP (***S**ecurity **SMTP***)	465		
POP3 (***P**ost **O**ffice **P**rotocol*),	110	TCP	1725, 1939

Serviciul distribuit	Port asociat	Socket	RFC de descriere
e-mail pentru workstations			
SPOP3 (***Security POP3***)	995		
FTP (*File Transfer Protocol*)	20	TCP	959, 2228-security extension
TFTP (***T**rivial **F**ile **T**ransfer **P**rotocol*)	69	UDP	1350
Telnet	23, *	TCP	854
SSH (***Security SH***)	22	TCP	1920
HTTP (***H**yper**T**ext **T**ransfer **P**rotocol*)	80	TCP	1945 (HTTP1.0), 2068 (HTTP1.1)
HTTPS HTTP over TLS (***T**ransport **L**ayer **S**ecurity*)	443		

4.2. Gestiunea adreselor Internet: InetAddress

Clasa reprezintă descrierea unei adrese Internet şi este folosită atât pentru comunicare prin UDP cât şi prin TCP. Prezentăm mai jos principalele metode ale acestei clase.

```
public final class InetAddress extends Object {

      public static InetAddress getLocalHost()
                                    throws UnknownHostException;
      public static syncronized InetAddress getByName(String host)
                                    throws UnknownHostException;
      public static syncronized InetAddress[] getAllByName
                                          (String host)
                                    throws UnknownHostException;

      public String getHostName();
      public byte[] getAddress();
}
```

Această clasă este finală, nu are constructor public, în schimb are trei metode statice care întorc instanţieri de clase `InetAddress`.

Metoda `getLocalHost`

Întoarce o `InetAddress` pentru host-ul local. Metoda `getByName` întoarce o `InetAddress` pentru maşina indicată prin `host`, care este fie o adresă `Internet`, fie o adresă IP în notaţia punctuală. Metoda

`getAllByName` întoarce un tablou de obiecte `InetAddress` cu toate adresele care localizează `host`.

Metoda `getHostName`

Întoarce numele host-ului gazdă, iar `getAddress` întoarce adresa IP în formă binară, într-un şir de 4 (patru) octeți primul fiind cel mai semnificativ (reprezentare *bigendian*).

În secțiunile următoare vom prezenta exemple de utilizare a acestei clase.

4.3. Transmitera si recepționarea prin UDP

Comunicarea prin UDP este adesea folosită în comunicații simple la care este mai importantă viteza şi mai rar în cazurile când datele ce se transmit sunt importante. Utilizarea acestei comunicări în Java este asigurată de două clase: `DatagramPacket` şi `DatagramSocket`.

4.3.1. Clasa DatagramPacket

Pachetele trimise prin protocolul UDP sunt descrise de clasa `DatagramPacket`, care gestionează structura pachetelor UDP, aşa cum sunt ele definite în standarde [48]. Structura acestei clase, cu principalele metode, este următoarea:

```
public final class DatagramPacket extends Object {

      public DatagramPacket(byte[] t, int lungt);
      public DatagramPacket(byte[] t, int lungt,
                            InetAddress adresa, int port);

      public byte[] getData();
      public int getLength();
      public int getPort();
      public InetAddress getAddress();
}
```

Primul constructor este folosit pentru recepția unei datagrame. De aceea, lui i se precizează un tablou de byte `t` şi o lungime maximă de recepționat `lungt`. Al doilea constructor este folosit pentru emisia unei datagrame. Se indică un tablou `t` de unde să se preia datele, lungimea `lungt` de transmis, `InetAddress` a destinatarului şi `port` la care acesta aşteaptă.

Cele patru metode sunt utile mai ales după recepția unui pachet:

- Metoda `getData` întoarce în tabloul de byte datele trimise/ recepționate de pachet.
- Metoda `getLength` întoarce lungimea informației utile trimise/ recepționate în pachet.
- Metoda `getPort` întoarce numărul portului de la care a emis expeditorul mesajului sau la care a primit destinatarul.
- Metoda `getInetAddress` întoarce un obiect de tip `InetAddress` care conține adresele Internet și IP ale expeditorului /receptorului.

4.3.2. Clasa DatagramSocket

Structura acestei clase este:

```
public class DatagramSocket extends Object {

      public DatagramSocket() throws SocketException;
      public DatagramSocket(int port) throws SocketException;

      public void send(DatagramPacket p) throws IOException;
      public syncronized void receive(DatagramPacket p)
                                          throws IOException;

      public getLocalPort();
      public syncronized void close();
}
```

Primul constructor creează un obiect `DatagramSocket` cu ajutorul căruia se pot trimite pachete de pe orice port local disponibil.

Metoda `send`

Transmite pachetul specificat ca argument. Adresa și portul destinație trebuie să fie deja trecute în pachet.

Al doilea constructor creează un obiect `DatagramSocket` care așteaptă la `port` primirea de pachete. Pentru a recepționa un pachet, trebuie creat un `DatagramPacket` cu un buffer pentru recepția mesajelor, dimensionat așa încât pachetul primit să încapă. În caz contrar ultimii octeți se vor pierde.

Metoda `receive`

Așteaptă primirea unui pachet pe care îl depune în obiectul `DatagramPacket` specificat ca argument.

Metoda `getLocalPort`

Este utilă pentru socket-ul care trimite pachete de la un port ales aleator de nucleul sistemului de operare local.

4.3.3. Exemple de comunicații prin UDP

4.3.3.1. Un talk simplu

Programele care urmează permit purtarea unui dialog între doi protagoniști, prin schimburi de pachete UDP, în maniera în care o face `talk`. Spre deosebire de `talk`, aici discuția se face disciplinat, unul dintre protagoniști trimite o linie, așteaptă să primească linia de răspuns de la celălalt, apoi îi trimite iarăși o linie ș.a.m.d. Cititorul poate ușor să extindă aceste programe ca să poată realiza un `talk` veritabil: este suficient ca la fiecare capăt să creeze două thread-uri, unul care scrie și altul care citește. Mai mult, prin niște interfețe grafice potrivite, împreună cu niște applet-uri, poate realiza un veritabil `chat`.

Pentru fixarea ideilor, să presupunem că avem doi protagoniști numiți `Activ` și `Pasiv`. Protagonistul `Pasiv` așteaptă de la `Activ` un prim mesaj care să pornească dialogul. Protagonistul `Activ` începe dialogul și tot el termină activitatea celor doi protagoniști atunci cînd transmite un mesaj care începe cu `STOP`.

Iată mai jos un exemplu de astfel de dialog. În fig. 4.2 este fereastra `Pasiv`, lansată pe mașina `nessie`, iar în fig. 4.3 este fereastra `Activ`, lansată pe mașina `florin`. După schimbul a două perechi de mesaje, `Activ` comandă `STOP`. Pentru o urmărire mai interesantă a dialogului, fiecare linie se încheie cu două ștampile de timp: ale expeditorului și ale destinatarului. (Diferențele orare nu țin de durata schimbului de pachete, ci de faptul că cele două mașini nu au ceasurile sincronizate!)

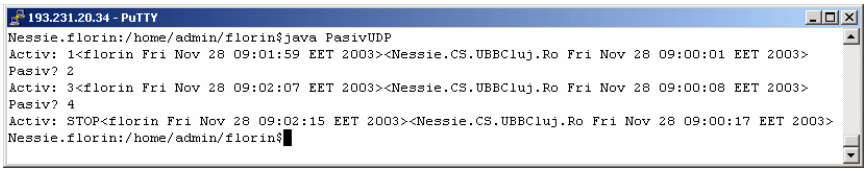

Figura 4.2. Fereastra `Pasiv`

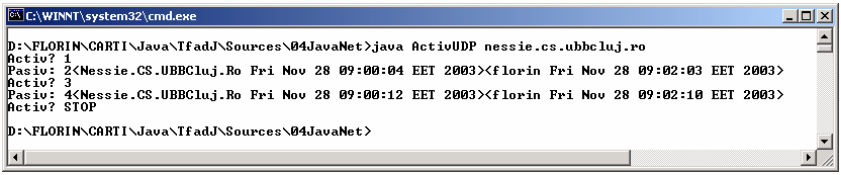

Figura 4.3. Fereastra `Activ`

Implementarea este realizată cu ajutorul a trei clase. Clasa `semnaturaTemporala` creează un string care conține numele mașinii locale, data și ora exactă a mașinii locale. Cu ajutorul acestei clase sunt realizate ștampilele de timp. Sursa acestei clase este prezentată în programul 4.1.

```
import java.net.*;
import java.util.*;

public class semnaturaTemporala {
   public String semnaturaTemporala() {
      String buf=null;
      try {
         //obtine numele masinii locale
         String hostname =
              (InetAddress.getLocalHost()).getHostName();
         buf = hostname+" "+
         //  Thread.currentThread().getThreadGroup()+"\t"+
         //  Thread.currentThread()+"\t"+
         new Date();
      } // try
      catch (UnknownHostException e) {
         e.printStackTrace();
      } // catch
      return buf;
   } // semnaturaTemporala.semnaturaTemporala
} // semnaturaTemporala
```

Programul 4.1. `semnaturaTemporala.java`

La crearea string-ului `hostname` se vede o primă utilizare a clasei `InetAddress`. Prin `getLocalHost` se obține obiectul `InetAddress` al mașinii locale, iar apoi prin `getHostName` se obține numele (adresa Internet) a mașinii locale.

Celelalte două programe au foarte multe elemente similare. Prezentăm mai întâi sursele acestora (programele 4.2 și 4.3), după care venim cu comentarii la ambele.

```
import java.io.*;
import java.net.*;

public class PasivUDP {

   public static void main(String[] a) throws Exception {
      semnaturaTemporala st = new semnaturaTemporala();
      BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
      DatagramSocket si = new DatagramSocket(2345); // Receptie la portul
                                                    // local 2345
      DatagramSocket so = new DatagramSocket(); // Emisie
      DatagramPacket p;
      InetAddress activ;
      byte [] b;
      String linie;
      for ( ; ; ) { // ciclu de asteptare mesaje
         b = new byte[1000];
         p = new DatagramPacket(b, b.length); // Creeaza p cu 1000 locuri
         si.receive(p); // Recepteaza linie de la portul local 23435
         activ = p.getAddress(); // De la cine a venit?
         linie = new String(p.getData()); // Converteste p la string
         linie = linie.substring(0,p.getLength()); // Retine doar ce a
                                                   // primit
         System.out.println("Activ: "+linie+"<"+
                            st.semnaturaTemporala()+">");
         if (linie.indexOf("STOP")==0)
            break;
         System.out.print("Pasiv? "); // Cere o linie
         linie = in.readLine(); // Citeste linia
         linie += "<"+st.semnaturaTemporala()+">";
         b = linie.getBytes(); // Converteste linia la bytes
         p = new DatagramPacket(b, b.length, activ, 5432);
         // Pune linia, adresa si portul destinatar
         so.send(p); // Trimite pachetul spre portul 5432 al lui Activ
      } // for
      so.close();
      si.close();
      in.close();
   } // PasivUDP.main
} // PasivUDP
```

Programul 4.2. `PasivUDP.java`

```
import java.io.*;
import java.net.*;

public class ActivUDP {

   public static void main(String[] a) throws Exception {
      semnaturaTemporala st = new semnaturaTemporala();
      BufferedReader in = new BufferedReader(
```

```
      new InputStreamReader(System.in));
      DatagramSocket si = new DatagramSocket(5432); // Receptie la portul
                                                    // local 5432
      DatagramSocket so = new DatagramSocket(); // Emisie
      DatagramPacket p;
      InetAddress pasiv =
         InetAddress.getByName((a.length==0)?"localhost":a[0]);
      byte [] b;
      String linie;
      for ( ; ; ) { // ciclu de asteptare mesaje
         System.out.print("Activ? "); // Cere o linie
         linie = in.readLine(); // Citeste linia
         linie += "<"+st.semnaturaTemporala()+">";
         b = linie.getBytes(); // Converteste linia la bytes
         p = new DatagramPacket(b, b.length, pasiv, 2345); // Pune
                                                  // linia, adresa si portul destinatar
         so.send(p); // Trimite pachetul spre portul 2345 al lui Pasiv
         if (linie.indexOf("STOP")==0)
            break;
         b = new byte[1000];
         p = new DatagramPacket(b, b.length); // Creeaza p cu 1000 locuri
         si.receive(p); // Recepteaza linie de la portul local 5432
         linie = new String(p.getData()); // Converteste p la string
         linie = linie.substring(0,p.getLength()); // Retine doar ce a
                                                   // primit
         System.out.println("Pasiv: "+linie+"<"+
                            st.semnaturaTemporala()+">");
      } // for
      so.close();
      si.close();
      in.close();
   } // ActivUDP.main
} // ActivUDP
```

Programul 4.3. **`ActivUDP.java`**

Ambele citesc linii de la intrarea standard prin intermediul obiectului `in`. Prompterul `Activ?`, respectiv `Pasiv?` cere câte o nouă linie. La capătul celălalt, linia transmisă este prefixată cu `Pasiv:` respectiv `Activ:` şi este urmată de ştampilele de timp ale celor doi protagonişti.

Fiecare dintre ele utilizează câte două socket:

- `si` care aşteaptă pachete la un anumit port - 2345 la `Pasiv` şi 5432 la `Activ`;
- `so` pentru trimiterea de pachete către celălalt.

Pentru emisie ambele transformă `String` - ul linie într-un şir de `bytes` pe care îl depune în pachetul `p`, în care mai pune adresa şi portul destinatarului.

Pentru recepție, ambele rezervă câte un pachet de 1000 octeți, din care extrage şi transformă în `String` câți octeți a primit efectiv. După recepție, fiecare își extrage din pachetul `p` adresa expeditorului, într-un obiect `InetAddress`.

4.3.3.2. Consultarea unui server time

Programul care urmează este un client care consultă serviciul standard `time` aflat pe un server care are activ acest serviciu. După cum am arătat mai sus, acest serviciu ascultă la portul 37, atât prin TCP cât şi prin UDP. Pentru moment, tratăm problema pentru cazul UDP.

Dacă se consultă RFC868, se va vedea că protocolul după care un client poate consulta timpul server-ului este:

- *Server-ul*:

  Aşteaptă să fie contactat la portul 37.

- *Clientul*:

  Trimite server-ului un pachet vid la portul 37.

- *Server-ul*:

  Recepționează acest pachet vid, reține adresa clientului şi portul local de la care i-a trimis clientul pachetul vid.

- *Server-ul*:

  Trimite clientului, la adresa şi portul reținute mai sus, un pachet cu patru octeți. În aceşti patru octeți server-ul pune, în conformitate cu calendarul şi ceasul lui, numărul de secunde scurse de la 01.01.1900 ora 00:00:00 şi până la momentul curent. Cei patru octeți conțin reprezentarea în ordinea *bigendian*, a acestui număr de secunde, priviți ca şi un întreg fără semn.

- *Clientul*:

  Recepționează numărul de secunde.

Proiectantul unui client `time` scris în Java trebuie să aibă în vedere două lucruri:

1. Clasa `Date` din pachetul `java.util` are un constructor care creează un obiect de tip `Data` plecând de la numărul de milisecunde scurs de la 01.01.1970 ora 00:00.

2. Java, în mod implicit, nu ştie să lucreze cu întregi fără semn. De exemplu, tipul de date `byte` are plaja de reprezentare [-128, 127], în cod complementar.

1. Primul neajuns se rezolvă simplu: între cele două date s-au scurs `2208988800` secunde (cine nu crede să calculeze!).

2. Al doilea neajuns se poate rezolva fie citind cei patru octeți cu metoda `readUnsignedByte` de la un obiect de tip `DataInputStream`, fie convertind cei patru octeți din pachet la un obiect `ByteArrayInputStream` care priveşte fiecare octet ca un număr între 0 şi 255. Indiferent de procedeu, conversia celor patru octeți la un întreg lung se face prin schema lui Horner, deoarece Java NU permite suprascrierea unei zone de memorie, aşa cum permite C sau C++ prin construcția `union`! În 2.4.3.2, programul 2.9, am prezentat o bibliotecă ce realizează astfel de conversii.

În metodele de consultare a timpului care urmează vom realiza conversiile în mod diferit, pentru varietatea expunerii. În prima metodă folosim metoda `ConvertBytes.getInt` a bibliotecii 2.9, în a doua metodă vom folosi direct un `ByteArrayInputStream`, iar în a treia vom converti folosind `readUnsignedByte`. Evident, cititorul este liber să folosească oricare modalitate de conversie doreşte.

Pentru a nu mai reveni încă odată asupra consultării timpului, prezentăm un program de consultare a unui server `time` care are trei metode:

- `timeUDP1` care foloseşte un singur socket şi un singur pachet pentru consultarea timpului (şi converteşte prin `ConvertBytes.getInt`);
- `timeUDP2`, puțin mai "complicat" din rațiuni didactice, care foloseşte pentru consultare două socket-uri şi două pachete, câte unul pentru fiecare sens de comunicație (şi converteşte prin `ByteArrayInputStream`);
- `timeTCP` care foloseşte conexiunea TCP care va fi explicată într-o secțiune următoare (şi converteşte prin `readUnsignedByte`),

În consecință vom folosi metoda `ConvertBytes.getInt` a acestei biblioteci. Sursa este dată în programul 4.4.

```
import java.io.*;
import java.net.*;
import java.util.*;
```

```
public class Time {
   // A se vedea RFC 868
   final long Sec_1900_1970 = 2208988800L; // numarul de secunde intre 1900
                                           // (RFC868) si 1970 (java.util)

   String timeUDP1(String host) throws Exception {
      DatagramSocket s;
      DatagramPacket p;
      Date c;
      long l;
      String h;
      byte [] b = new byte[4];
      p = new DatagramPacket(b, 0, InetAddress.getByName(host),
                             37);
      s = new DatagramSocket();
      s.send(p);                    // Trimite la server un pachet vid (RFC868)
      p.setLength(4);
      s.receive(p);                 // Asteapta cei 4 octeti si-i pune in pachet
      h = (p.getAddress()).getHostName(); // Adresa masinii de unde a venit
                                          // pachetul se scoate din pachet
      s.close();
      Integer[] in = new Integer[1];
      ConvertBytes.getInt(p.getData(), 0, in);
      l = (long)in[0].intValue(); // l contine numarul de secunde de la 01:01:1900
      l -= Sec_1900_1970;    // Numarul de secunde pana la 1970
      c = new Date(l*1000); // Creaza o data in milisecunde de la 1970
      return c.toString()+" "+h;
   } // Time.timeUDP1

   String timeUDP2(String host) throws Exception {
      DatagramSocket so, si;
      DatagramPacket po, pi;
      Date c;
      int portLocal;
      long l;
      String h;
      byte [] b = new byte[4];
      po = new DatagramPacket(b, 0, InetAddress.getByName(host),
                              37);
      pi = new DatagramPacket(b, 4);
      so = new DatagramSocket();
      portLocal = so.getLocalPort();
      so.send(po);                  // Trimite la server un pachet vid (RFC868)
      so.close();
      si = new DatagramSocket(portLocal);
      si.receive(pi);               // Asteapta cei 4 octeti si-i pune in pachet
      h = (pi.getAddress()).getHostName(); // Adresa masinii de unde a venit
                                           // pachetul se scoate din pachet
      si.close();
      Integer[] in = new Integer[1];
      ConvertBytes.getInt(pi.getData(), 0, in);
      l = (long)in[0].intValue(); // l contine numarul de secunde de la 01:01:1900
      l -= Sec_1900_1970;    // Numarul de secunde pana la 1970
```

```
      c = new Date(l*1000); // Creaza o data in milisecunde de la 1970
      return c.toString()+" "+h;
   } // Time.timeUDP2

   String timeTCP(String host) throws Exception {
      Socket s;
      DataInputStream in;
      Date c;
      long l;
      String h;
      s = new Socket(host, 37);
      h = (s.getInetAddress()).getHostName(); // Se stie deja adresa masinii
                                              // server time
      in = new DataInputStream(s.getInputStream()); // Pe in vin 4 octeti in
                                                    // bigendian
      byte[] b = new byte[4];
      in.read(b);
      Integer[] i = new Integer[1];
      ConvertBytes.getInt(b, 0, i);
      l = (long)i[0].intValue(); // l contine numarul de secunde de la 01:01:1900
      in.close();
      s.close();
      l -= Sec_1900_1970;          // Numarul de secunde pana la 1970
      c = new Date(l*1000);        // Creaza o data in milisecunde de la 1970
      return c.toString()+" "+h;
   } // Time.timeTCP

   public static void main(String[] a) throws Exception {
      Time t = new Time();
      System.out.println(t.timeUDP1("linux.scs.ubbcluj.ro"));
         // Trebuie sa fie server time si sa asculte UDP

      System.out.println(t.timeUDP2("linux.scs.ubbcluj.ro"));
         // Trebuie sa fie server time si sa asculte UDP

      System.out.println(t.timeTCP("linux.scs.ubbcluj.ro"));
         // Trebuie sa fie server time si sa asculte TCP

      System.out.println((new Date())+
            " "+(InetAddress.getLocalHost()).getHostName());
   } // Time.main
} // Time
```

Programul 4.4. `Time.java`

Fiecare dintre cele trei metode are ca parametru de intrare adresa maşinii care este server de `time`. Metoda `main` le apelează pe celelalte trei, după care afişează şi ora maşinii locale (a se vedea desincronizarea ceasurilor).

4.3.3.3. Un alt exemplu de comunicație prin UDP

Exemplul care urmează este unul simplu:

Un program server aşteaptă mesaje de la clienți şi le afişează pe ieşirea standard împreună cu adresa şi portul de expediție.

Programul a fost publicat în [10]. Singura modificare făcută a fost folosirea unor metode Java nedepăşite (*non deprecation*) pentru conversia `String` <-> tablou de `bytes`.

Programul server `DatagramRecv` aşteaptă să primească un mesaj de maximum 1024 caractere de la un client oarecare pe portul 9898. Mesajul este primit în spațiul rezervat `pachet`. După primire, mesajul este afişat pe ieşirea standard a server-ului, împreună cu adresa clientului expeditor şi cu portul pe care s-a expediat. Sursa acestui server este prezentată în programul 4.5.

```
import java.io.*;
import java.net.*;

// Receptioneaza un mesaj prin UDP:
// Textul mesajului este primit prin socket este afisat la iesire

public class DatagramRecv {
  static final int PORT = 9898;
  public static void main (String args[]) throws Exception {

    byte[] b = new byte[1024];
    String mesaj;

    // Creaza un pachet gol unde sa receptioneze
    DatagramPacket pachet = new DatagramPacket(b, b.length);

    // Creaza socket care asteapta la port
    DatagramSocket s = new DatagramSocket(PORT);
    for(;;) {
      // Asteapta sosirea unei datagrame
      s.receive(pachet);

      // Converteste continutul la un string
      mesaj = new String(b, 0, pachet.getLength());

      // Afiseaza mesajul la iesire
      System.out.println("Receptionat de la: "+
        pachet.getAddress().getHostName()+":"+
        pachet.getPort()+":\n"+mesaj);
    }
  }
}
```

Programul 4.5. **`DatagramRecv.java`**

Sursa clientului `DatagramSend` este prezentată în programul 4.6.

```
import java.io.*;
import java.net.*;

// Trimite un mesaj prin UDP:
// Masina destinatie este primul argument al liniei de comanda
// Textul mesajului este al doilea argument al liniei

public class DatagramSend {
  static final int PORT = 9898;
  public static void main (String args[]) throws Exception {
    if (args.length !=2) {
      System.out.println("Utilizare java Masina Mesaj");
      System.exit(1);
    }

    // Obtine adresa Internet a masinii
    InetAddress adresa = InetAddress.getByName(args[0]);

    // Converteste mesajul intr-un sir de bytes
    int lungime = args[1].length();
    byte [] mesaj = args[1].getBytes();

    // Initializeaza pachetul cu date si adresa
    DatagramPacket pachet = new DatagramPacket(
                    mesaj, lungime, adresa, PORT);

    // Creaza socket si emite pachetul prin el
    DatagramSocket s = new DatagramSocket();
    s.send(pachet);
  }
}
```

Programul 4.6. `DatagramSend.java`

4.4. Comunicare prin TCP

4.4.1. Clasa Socket

Obiectele de tip `Socket` sunt folosite atât la nivel de client, cât şi la nivel de server, pentru comunicare prin TCP cu partenerul. Structura acestei clase este:

```
public final class Socket extends Object {

      public Socket(String host, int port)
                        throws UnknownHostException, IOException;
      public Socket(InetAddress adresa, int port)
```

```
                        throws IOException;

        public InputStream getInputStream() throws IOException;
        public OutputStream getOutputStream() throws IOException;

        public int getLocalPort();
        public int getPort();
        public InetAddress getInetAddress();

        public syncronized void close() throws IOException;
}
```

La crearea unui socket, adresa maşinii la distanţă poate fi precizată fie printr-un `String` ce conţine adresa IP sau Internet, fie printr-un obiect `InetAddress` pregătit în prealabil. În plus, se indică şi portul de comunicaţie.

Metodele `getInputStream` şi `getOutputStream`

Sunt cele mai importante. Ele întorc obiecte de tip `InputStream` respectiv `OutputStream`, prin intermediul cărora se realizează schimburile de octeţi între parteneri. În legătură cu aceste obiecte, trebuie să facem ***o precizare importantă***: metodele de tip `read` (citiri binare cu şi fără interpretare) ale acestor obiecte nu asigură citirea neapărată a tuturor octeţilor de pe socket! De fapt, metoda `read` întoarce numărul de octeţi pe care a reuşit să-i citească efectiv!. De aceea, programatorul trebuie să aibă grijă să repete citirea de pe socket atunci când nu a primit numărul de octeţi solicitat.

Metoda `getLocalPort`

Întoarce numărul portului local folosit de socket.

Metodele `getPort` şi `getInetAddress`

Întorc respectiv portul de la distanţă la care este conectat socket-ul, respectiv obiectul `InetAddress` al partenerului de la distanţă.

Aplicaţiile care comunică la distanţe mari, ca şi aplicaţiile complexe şi sofisticate utilizează transmisia prin TCP. Perechile client-server pot fi ambele scrise în Java, dar este posibil, după cum vom vedea printr-un exemplu în secţiunea următoare, ca unii dintre protagonişti să fie scrişi în alt limbaj.

4.4.2. Clasa ServerSocket

Obiectele de tip `ServerSocket` au rolul de a aştepta şi de a accepta cererile de conexiune de la clienți. Server-ul e înştiințat de stabilirea conexiunii prin primirea unui obiect de tip `Socket`, prin intermediul căruia se va realiza comunicația. Structura clasei `ServerSocket` este următoarea:

```
public final class ServerSocket extends Object {

      public ServerSocket(int port) throws IOException;
      public ServerSocket(int port, int nr) throws IOException;

      public Socket accept() throws IOException;

      public int getLocalPort();
      public InetAddress getInetAddress();
      public void close();
}
```

Se observă că, la crearea unui obiect `SeverSocket`, se precizează doar portul de comunicație. Dacă el are valoarea 0, atunci se consideră orice port liber. Parametrul `nr` din al doilea constructor stabileşte lungimea maximă a cozii cererilor de conexiune. Dacă o cerere vine într-un moment în care coada este plină, conexiunea e refuzată.

Metoda `accept`

Este cea mai importantă. În momentul conectării unui client, aceasta întoarce un socket prin intermediul căruia se realizează comunicația.

Exemplele din secțiunile următoare vor clarifica modul de comunicare TCP din Java.

4.5. Exemple de comunicații TCP

4.5.1. Un talk simplu (TCP)

Reluăm problema dialogului între doi protagonişti: `Pasiv` şi `Activ`, problemă enunțată de noi în 4.3.3.1 şi implementată prin UDP. Exact aceeaşi funcționalități le vom realiza prin comunicare TCP. Pentru aceasta, protagonistul `Pasiv` va avea rolul de server, iar `Activ` rolul de client.

Cele două surse sunt prezentate în programele 4.:

```
import java.io.*;
import java.net.*;

public class PasivTCP {

   public static void main(String[] a) throws Exception {
      semnaturaTemporala st = new semnaturaTemporala();
      BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
      ServerSocket ss = new ServerSocket(2345);
      Socket s;
      BufferedReader sin;
      PrintStream sout;
      String linie;
      for ( ; ; ) { // ciclu de asteptare mesaje
         s = ss.accept(); // S-a stabilit o conexiune
         sin = new BufferedReader(
                        new InputStreamReader(s.getInputStream()));
         sout = new PrintStream(s.getOutputStream());
         for ( ; ; ) { // Ciclul de dialog in cadrul conexiunii
            linie = sin.readLine();
            System.out.println("Activ: "+linie+"<"+
                                 st.semnaturaTemporala()+">");
            if (linie.indexOf("STOP")==0)
               break;
            System.out.print("Pasiv? "); // Cere o linie
            linie = in.readLine();       // Citeste linia
            linie += "<"+st.semnaturaTemporala()+">";
            sout.println(linie);         // Trimite linia la client
         } // for conexiune
         break;
      } // for accept
      sout.close();
      sin.close();
      in.close();
      s.close();
      ss.close();
   } // PasivTCP.main
} // PasivTCP
```

Programul 4.7. `PasivTCP.java`

```
import java.io.*;
import java.net.*;

public class ActivTCP {

   public static void main(String[] a) throws Exception {
      semnaturaTemporala st = new semnaturaTemporala();
      BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
      Socket s = new Socket((a.length==0)?"localhost":a[0], 2345);
      BufferedReader sin = new BufferedReader(
```

```
                    new InputStreamReader(s.getInputStream()));;
      PrintStream sout = new PrintStream(s.getOutputStream());
      String linie;
      for ( ; ; ) { // ciclu de asteptare mesaje

         System.out.print("Activ? "); // Cere o linie
         linie = in.readLine(); // Citeste linia
         linie += "<"+st.semnaturaTemporala()+">";
         sout.println(linie); // Trimite linia
         if (linie.indexOf("STOP")==0)
            break;
         linie = sin.readLine();
         System.out.println("Pasiv: "+linie+"<"+
                        st.semnaturaTemporala()+">");
      } // for
      sout.close();
      sin.close();
      in.close();
      s.close();
   } // ActivTCP.main
} // ActivTCP
```

Programul 4.8. `ActivTCP.java`

Urmărind programul server, trebuie remarcat faptul că metoda `accept` întoarce, în momentul conectării unui client, un obiect `Socket`. Acesta din urmă este folosit pentru schimbul efectiv de octeţi dintre client şi server. Momentul conectării este momentul în care clientul creează obiectul socket cu precizarea adresei server-ului şi a portului la care acesta aşteaptă.

Apoi mai trebuie remarcat faptul că din obiectele socket, atât cel de la client cât şi cel de la server, se creează câte o pereche de obiecte pentru schimbul în cele două sensuri. Plecând de la obiectele întoarse de metodele `getInputStream` şi `getOutputStream`, în acest exemplu am construit perechile de obiecte `BufferedReader` şi `PrintStream`, care permit efectuarea uşoară de schimburi de linii (tip `String`).

Mai trebuie remarcat faptul că spre deosebire de UDP, aici se utilizează explicit numai un singur port, cel de la server. De asemenea, schimbul de linii între protagonişti se face ca şi cum s-ar efectua operaţii I/O locale!

Lăsăm pe seama cititorului să compare cele două implementări şi să constate că implemetarea TCP este mult mai simplă.

4.5.2. O aplicaţie rezumat la distanţă: RemoteDir

Problema care se pune este [10]:

Clientul îi transmite server-ului numele unui director, presupus a fi pe server. Server-ul execută rezumatul acestui director şi îl transmite ca răspuns clientului.

Vom implementa un server scris în Java şi trei clienţi: unul Java standalone, un applet Java şi un client scris în C. Server-ul aşteaptă să fie contactat la portul 8899.

4.5.2.1. Programul ServerRemoteDir

Programul 4.9 prezentă sursa serverului.

```
import java.io.*;
import java.net.*;

public class ServerRemoteDir extends Thread {
   public final static int PORT = 8899;
   protected ServerSocket socket_intalnire;
   // Creeaza socket-ul de intalnire si initializeaza thread.
   public ServerRemoteDir() {
      try {
         socket_intalnire = new ServerSocket(PORT);
      } // try
      catch (Exception e) {
         e.printStackTrace();
         System.exit(1);
      } // catch
      this.start();
   } // ServerRemoteDir.ServerRemoteDir

   // Corpul thread-ului server. La fiecare conectare se creaza
   // un obiect de tip Conectare (definit mai jos), care trateaza
   // conexiunea printr-un socket nou
   public void run() {
      Socket socket_client;
      try {
         for (;;) {
            socket_client = socket_intalnire.accept();
            Connectare c = new Connectare(socket_client);
         } // for
      } // try
      catch (Exception e) {
         e.printStackTrace();
      } // catch
   } // ServerRemoteDir.run

   // Initializare server
   public static void main(String[] a) {
      new ServerRemoteDir();
   } // ServerRemoteDir.main
```

```
} // ServerRemoteDir
```

Programul 4.9. `ServerRemoteDir.java`

Constructorul `ServerRemoteDir` creează doar `server_intalnire` prin care se aşteaptă contactarea de către clienți, la portul 8899, fixat prin constanta `PORT`.

Partea curentă a acestui thread (metoda `run`) aşteaptă prin `accept` contactarea de către un client. În momentul conectării se creează un thread nou, obiect instanță a clasei `Conectare`. Pentru instanțiere se transmite obiectul socket obținut la conectare.

Sursa thread-ului care comunică direct cu clientul care s-a conectat este dată în programul 4.10.

```
import java.io.*;
import java.net.*;

// Clasa care defineste thread-ul de tratare a unei conexiuni
class Conectare extends Thread {
   protected DataInputStream in;
   protected DataOutputStream out;
   Socket client;

   // Initializeaza "fisierul" de socket si thread-ul
   public Conectare(Socket socket_client) {
      client = socket_client;
      try {
         in = new DataInputStream(client.getInputStream());
         out = new DataOutputStream(client.getOutputStream());
      } // try
      catch (Exception e) {
         e.printStackTrace();
         return;
      } // catch
      this.start();
   } // Conectare.Conectare

   // Furnizeaza serviciul
   public void run() {
      final String NL = System.getProperty("line.separator");
      try {
         // Citeste lungimea si numele directorului
         int nr_oct = in.readInt();
         System.out.println(nr_oct);
         byte[] b = new byte[nr_oct];
         in.read(b);
         String director = new String(b);
         System.out.println(director);

         File f = new File(director);
```

```
          if (f.isDirectory()) {
              // Obtine rezumatul si-l face un string unic
              String[] rezumat = f.list();
              String s = new String();
              for(int i=0; i<rezumat.length; i++)
                 s += rezumat[i]+NL;
              b = s.getBytes(); // Converteste in sir de bytes
              // Trimite lungimea si continutul
              out.writeInt(s.length());
              System.out.println(s.length());

              out.write(b);
           } // if director
          else {
              String s = new String();
              String s=director+ " nu este un director";
              out.writeInt(s.length());
              b=s.getBytes();
              out.write(b);
            }
// in cazul in care cel de-al doilea parametru nu este director se
// intra intr-o asteptare infinita (daca nu se pune varianta de else)
// clientul asteapta infinit rezumatul de la server iar serverul nu // trimite nimic

        } // try
        catch (Exception e) {
           e.printStackTrace();
        } // catch
    } // Conectare.run
} // Conectare
```

Programul 4.10. `Conectare.java`

Partea de constructor `Conectare` se rezumă la crearea obiectelor `in` şi `out` de tipuri `DataInputStream` şi respectiv `DataOutputStream` prin care thread-ul comunică cu clientul.

Partea de acțiune efectivă este executată de metoda `run` a acestei clase. Aceasta, prin intermediul obiectului `f` de tip `File` obține rezumatul directorului solicitat. După obținere, acesta este transmis clientului.

Comunicarea dintre server şi client este organizată puțin mai special. Orice mesaj este compus dintr-un întreg şi un şir de caractere ASCII. Întregul precizează numărul total de caractere din şir. Trebuie precizat că nu se transmite un `String` în sens Java, ci doar coduri `ASCII`. Din această cauză, înainte de transmisie string-ul este transformat în tablou de octeți. La recepție se execută operația inversă, adică transformarea şirului de `byte` în string Java.

Am ales această soluție din mai multe motive şi anume:

- Sistemele de operare afişează rezumatele de directori în ASCII şi nu în Unicode cum se reprezintă `String` în Java.

- Nu este obligatoriu ca ambii protagonişti (client şi server) să aibă ca limbaj gazdă Java.
- Din raţiuni de portabilitate sporită, majoritatea mesajelor din aplicaţiile de comunicaţii încep cu întregi care dau lungimea efectivă a mesajului.

4.5.2.2. Clientul standalone RemoteDir

Sursa clientului standalone `RemoteDir` este prezentată în programul 4.11.

```
import java.io.*;
import java.net.*;

public class RemoteDir {
   public static final int PORT = 8899;
   public static void main(String[] a) {
      Socket s;
      String rezumat;
      try {
         s = new Socket(a[0], PORT);
         DataInputStream in =
             new DataInputStream(s.getInputStream());
         DataOutputStream out =
             new DataOutputStream(s.getOutputStream());
         byte[] b = a[1].getBytes();   // Nume director in sir de bytes
         out.writeInt(a[1].length()); // Trimite lungimea
         out.write(b);                // Trimite numele de director
         int nr_oct = in.readInt();       // Citeste lungimea
         b = new byte[nr_oct];
         in.read(b);                      // Citeste rezumatul
         rezumat = new String(b);
         System.out.println(rezumat);     // Afiseaza rezumatul
      } // try
      catch (Exception e) {
         e.printStackTrace();
      } // catch
   } // RemoteDir.main
} // RemoteDir
```

Programul 4.11. `RemoteDir.java`

Clientul primeşte în linia de comandă adresa Internet a maşinii ca prim argument şi numele de director ca al doilea argument.

4.5.2.3. Clientul applet RemoteDirApp

A doua variantă de client este un applet. Sursa lui este prezentată în programul 4.12.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class RemoteDirApp extends Applet implements ActionListener
{
   public static final int PORT = 8899;
   Socket s;
   DataInputStream in;
   DataOutputStream out;
   TextField director;
   String masina;
   TextArea rezumat;
    // Linia "director" va primi numele directorului
    // String-ul "masina" va fi preluat ca parametru HTML
    // Zona de date "rezumat" va afisa rezumatul
   public void init() {
      try {
         director = new TextField(
             "Aici se da numele directorului");   // Preia numele de director
         director.addActionListener(this);
         rezumat = new TextArea();
         rezumat.setEditable(false);
         this.setLayout(new BorderLayout());
         this.add("North", director);
         this.add("Center", rezumat);
         masina = this.getParameter("Masina"); // Citeste parametrul HTML
         s = new Socket(masina, PORT);
         in = new DataInputStream(s.getInputStream());
         out = new DataOutputStream(s.getOutputStream());
      } // try
      catch (Exception e) {
         e.printStackTrace();
      } // catch
   } // RemoteDirApp.init

   // Cand se da la director un text, evenimentul este
   // interceptat si linia este trimisa serverului
   public void actionPerformed(ActionEvent e) {
      int n;
      byte[] b;
      if (e.getSource() == director) {
         try {
            b = (director.getText()).getBytes();
            out.writeInt(b.length); // Trimite lungimea
```

```
            out.write(b);              // Trimite numele de director
            n = in.readInt();
            b = new byte[n];
            in.read(b);
            rezumat.setText(new String(b));
         } // try
         catch (Exception e2) {
            ;
         } // catch
      } // if
   } // RemoteDirApp.actionPerformed
} // RemoteDirApp
```

Programul 4.12. **`RemoteDirApp.java`**

Clasa `RemoteDirApp` defineşte fereastra de acces prin intermediul căreia are loc dialogul dintre utilizator şi client. Deoarece aici nu am putut prelua parametrii din linia de comandă (applet-ul nu permite aşa ceva) şi nici nu putem afişa pe ieşirea standard, am adoptat o soluţie mixtă:

- adresa maşinii server este preluată printr-un parametru preluat din fişierul sursă HTML de lansare a clientului;
- numele de director este preluat printr-o componentă de tip `java.awt.TextField`;
- rezumatul directorului este afişat într-o fereastră de tip `java.awt.TextArea`.

Partea din fişierul HTML necesară apelului acestui applet este prezentată în programul 4.13.

```
<html>
      <head><title>RemoteDieApp</title></head>
<body>
      <APPLET code="RemoteDirApp.class" width=500 height=300>
         <PARAM name="Masina" value = "florin.cs.ubbcluj.ro">
      </APPLET>
</body>
</html>
```

Programul 4.13. **`RemoteDirApp.html`**

În cadrul metodei `init`, activă la lansarea applet-ului, se configurează mai întâi interfaţa grafică. Apoi este preluat numele maşinii din documentul HTML (tagul <PARAM). Urmează crearea obiectului socket şi a celor două obiecte de tip `DataInputStream` şi `DataOutputStream`, destinate schimbului dintre applet şi `ServerRemoteDir`.

În linia de sus a interfeței grafice afișate de navigator (fig.4.4), utilizatorul scrie numele de director dorit. În momentul în care se tastează <ENTER>, se creează un obiect eveniment şi este activată metoda `actionPerformed`. Aici se trimite server-ului lungimea numelui de director şi apoi numele propriu-zis, după care aşteaptă răspunsul de la server. Rezumatul răspuns este afişat în zona `TextArea` de pe interfață.

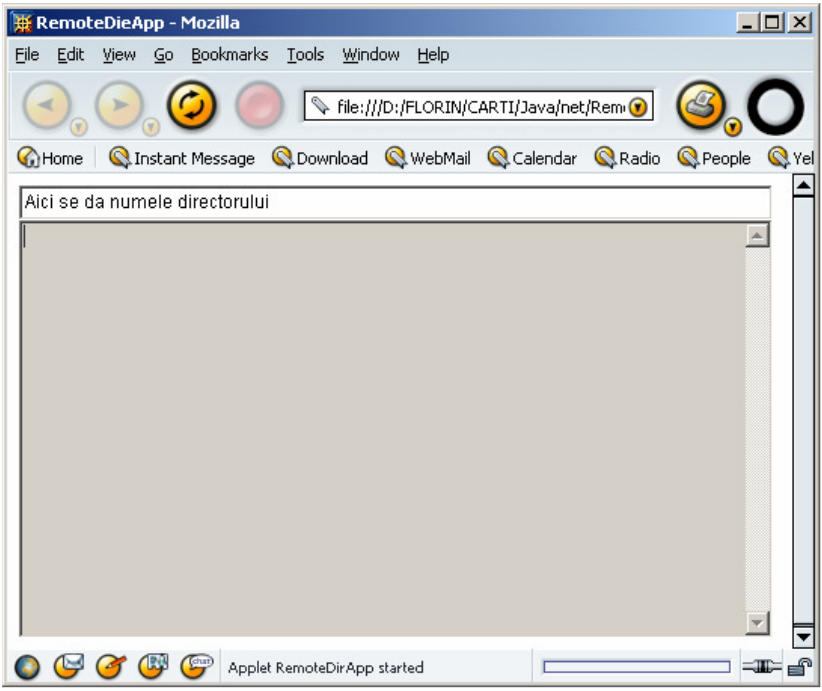

Figura 4.4. Interfața grafică generată de `RemoteDirApp`

4.5.2.4. Client RemoteDir scris în C

Al treilea client pentru `ServerRemoteDir` este elaborat în C sub Unix. Sursa acestui program este:

```
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include "ConvertBytes.cpp" // Pentru conversiile lungimilor!

#define PORT_SERVER 8899
#define DIRSIZE 8192

   main (int c, char *a[]) {
      char dir[DIRSIZE];
      int sd, i, l;
      struct sockaddr_in serv_addr;
      struct hostent *hp;
      sd = socket (AF_INET, SOCK_STREAM, 0);
      hp = gethostbyname (a[1]);
      memset ((char *) &serv_addr, 0, sizeof (serv_addr));
      serv_addr.sin_family = AF_INET;
      memcpy((char *) &serv_addr.sin_addr.s_addr,
                      hp->h_addr_list[0], hp->h_length);
      serv_addr.sin_port = htons (PORT_SERVER);
      connect (sd, (struct sockaddr *)
               &serv_addr,sizeof (serv_addr));

      addInt(strlen(a[2]), dir, 0); // Converteste la int Java
      send (sd, dir, 4, 0); // Trimite lungimea
      send (sd, a[2], strlen (a[2]), 0); // Trimite numele

      for (i=0; i<4; i++)
         recv (sd, &dir[i], 1, 0); // Citeste lungimea
      getInt(dir, 0, &l); // Converteste lungimea

      for (i=0; i<l; i++)
         recv (sd, &dir[i], 1, 0);
      printf ("%s\n", dir);
      close (sd);
   } // RemoteDirC.main
```

Programul 4.14. `RemoteDirC.c`

Din punct de vedere al comportamentului, acest client este identic cu cel standalone din Java (`RemoteDir`). Deci primul argument al liniei de comandă este adresa maşinii lui `ServerRemoteDir` iar al doilea argument este numele directorului. Pe lângă elementele specifice C pentru comunicații, la acest client trebuie să mai prezentăm două particularități:

1. conversia reprezentărilor binare, posibil diferite, a întregilor din C şi cei din Java;
2. citirea de pe socket caracter cu caracter.

1. În legătură cu conversia întregilor:

Se ştie că în Java întregii sunt reprezentați pe patru octeți şi în ordinea *bigendian.* Implementările întregilor din C depind de platformă. Pentru

conversii am folosit biblioteca C descrisă în 2.4.3, programul 2.10.

2. În legătură cu citirea de pe socket:

Aşa cum am precizat la descrierea clasei `Socket`, operaţia `read` nu asigură citirea tuturor octeţilor de pe socket! Platforma Java are grijă să-şi buffer-izeze datele aşa încât acest neajuns să fie transparent pentru program. Nu acelaşi lucru se poate spune despre C.

În consecinţă, am preferat să citim octet cu octet. Mai întâi am citit pe rând cei patru octeţi ai întregului lungime, iar după conversia reprezentării lungimii am citit, câte unul, restul octeţilor întorşi de server.

4.5.3. Folosirea ca partener a unor servicii standard

Orice sistem de operare actual (Unix, Windows etc.), aşa cum am arătat mai sus (în 4.1.2), are operaţionale o serie de servicii distribuite. Protocoalele acestora sunt de regulă independente de platformă, spre a putea comunica între ele. În cele ce urmează vom da cinci exemple de comunicaţii în care unul dintre parteneri (fie clientul, fie server-ul) este un serviciu standard, iar al doilea partener este un program Java.

4.5.3.1. Consultarea unui server time prin TCP

Problema a fost prezentată în detaliu la conectarea time prin UDP, 4.3.3.2. Prezentăm mai jos doar funcţia de obţinere a timpului prin TCP, copiată din programul 4.4.

```
String timeTCP(String host) throws Exception {
  Socket s;
  DataInputStream in;
  Date c;
  long l;
  String h;
  s = new Socket(host, 37);
  h = (s.getInetAddress()).getHostName(); // Se stie deja adresa
                                          // masinii server time
  in = new DataInputStream(s.getInputStream()); // Pe in vin 4 octeti in
                                                // bigendian
  byte[] b = new byte[4];
  in.read(b);
  Integer[] i = new Integer[1];
  ConvertBytes.getInt(b, 0, i);
  l = (long)i[0].intValue(); // l contine numarul de secunde de la 01:01:1900
  in.close();
```

```
      s.close();
      l -= Sec_1900_1970;          // Numarul de secunde pana la 1970
      c = new Date(l*1000);        // Creaza o data in milisecunde de la 1970
      return c.toString()+" "+h;
   } // Time.timeTCP
```

4.5.3.2. Autentificare folosind un server FTP

În multe aplicații se simte nevoia unui mecanism prin care un utilizator, de pe o anumită mașină, să fie autentificat printr-o parolă. De regulă, această activitate nu este una prea simplă şi în plus este relativ nesigură: este posibil ca un rău intenționat să "prindă" parola!

În cele ce urmează dăm o soluție extrem de simplă şi de elegantă de autentificare - lăsăm pe seama server-ului de FTP să autentifice! Evident, metoda funcționează numai dacă pe mașina respectivă există un server de FTP.

Invităm cititorul să consulte RFC959 spre a vedea cum funcționează FTP şi care este, exact, protocolul de acces la un server FTP. Noi am extras pentru autentificare partea de protocol aferentă:

- conectarea la portul 21;
- răspunsul la conectare;
- transmiterea spre server a unui nume de user (comanda `user`);
- răspunsul la această comandă;
- transmiterea parolei (comanda `pass`);
- răspunsul la această comandă.

Mesajele transmise spre server sunt linii ASCII cu sintaxe prestabilite, iar răspunsurile server-ului FTP sunt linii ASCII care, în caz de succes, încep cu câte un anumit număr.

Programul următor definește clasa `AuthFTP`, partea ei esențială fiind metoda statică (de clasă) `isFTPUser`. Acesta din urmă are trei parametri: adresa server-ului FTP, numele de utilizator şi parola acestuia. Metoda întoarce `true` sau `false`, după cum utilizatorul este autentificat sau nu.

Programul mai conține şi metoda `main` pentru a testa funcția `isFTPUser`. Sursa este prezentată în programul 4.15.

```
import java.io.*;
import java.net.*;

public class AuthFTP {
   public static boolean isFTPUser(String host, String user,
                                   String password) {
      Socket socket = null;
      BufferedReader in = null;
      PrintStream out = null;
```

```
      String s;
      try {
         socket = new Socket(host, 21);
         in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
         out = new PrintStream(socket.getOutputStream());
         s = in.readLine();
         if (s.indexOf("220") != 0)
            return(false);
         out.println("user "+user);
         s = in.readLine();
         if (s.indexOf("331") != 0)
            return(false);
         out.println("pass "+password);
         s=in.readLine();
         if (s.indexOf("230") != 0)
            return(false);
         return(true);
      } // try
      catch (Exception e) {
         e.printStackTrace();
         return(false);
      } // catch
      finally {
         try {
            if (out != null)
               out.close();
            if (in != null)
               in.close();
            if (socket != null)
               socket.close();
            in = null;
            out = null;
            socket = null;
         } // try
         catch (Exception e) {
         } // catch
      } // finally
   } // AuthFTP.isUser

   public static void main(String a[]) {
      if (AuthFTP.isFTPUser(a[0], a[1], a[2]))
         System.out.println("Bine");
      else
         System.out.println("Rau");
   } // AuthFTP.main
} // AuthFTP
```

Programul 4.15. `AuthFTP.java`

4.5.3.3. Autentificarea prin server POP3

O soluție alternativă de autentificare este aceea folosind server-ul POP3 (evident dacă acesta este operațional). Soluția este extrem de asemănătoare cu cea a lui AuthFTP. Invităm cititorul să consulte RFC1939 spre a vedea cum funcționează POP3. Partea de autentificare este extrem de asemănătoare cu cea a server-ului FTP. Singura deosebire constă în faptul că răspunsurile afirmative ale server-ului POP3 încep cu string-ul "+OK".

Sursa este prezentată în programul 4.16.

```
import java.io.*;
import java.net.*;

public class AuthPOP3 {
   public static boolean isPOP3User(String host, String user,
                                    String password) {
      Socket socket = null;
      BufferedReader in = null;
      PrintStream out = null;
      String s;
      try {
         socket = new Socket(host, 110);
         in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
         out = new PrintStream(socket.getOutputStream());
         s = in.readLine();
         if (s.indexOf("+OK") != 0)
            return(false);
         out.println("user "+user);
         s = in.readLine();
         if (s.indexOf("+OK") != 0)
            return(false);
         out.println("pass "+password);
         s=in.readLine();
         if (s.indexOf("+OK") != 0)
            return(false);
         return(true);
      } // try
      catch (Exception e) {
         e.printStackTrace();
         return(false);
      } // catch
      finally {
         try {
            if (out != null)
               out.close();
            if (in != null)
               in.close();
            if (socket != null)
               socket.close();
            in = null;
```

```
            out = null;
            socket = null;
         } // try
         catch (Exception e) {
         } // catch
      } // finally
   } // AuthFTP.isUser
   public static void main(String a[]) {
      if (AuthPOP3.isPOP3User(a[0], a[1], a[2]))
         System.out.println("Bine");
      else
         System.out.println("Rau");
   } // AuthPOP3.main
} // AuthPOP3
```

Programul 4.16. `AuthPOP3.java`

4.5.3.4. Un server de comenzi apelabil prin telnet

Unul dintre clienţii standard remarcabili este `telnet`. Pentru detalii, cititorul poate consulta RFC854, în care se descrie complet `telnet`. Acesta se lansează folosind comanda:

```
telnet adresaMasina [ port ]
```

Dacă `port` lipseşte, atunci se conectează pe portul 23 şi creează un client terminal, de la care se poate dialoga cu sistemul de operare prin comenzi specifice lui (sistemului de operare).

Acţiunea clientului la alte porturi este specifică portului/maşinii la care se face `telnet`.

În cele ce urmează vom crea un server, l-am numit `ServerTelnet`, la care clienţii sunt apeluri `telnet` pe portul 5656. Acţiunea tandemului client (`telnet`) - `ServerTelnet` este următoarea:

- Clientul transmite server-ului o linie care trebuie să fie o comandă de pe sistemul de operare pe care rulează server-ul.
- Server-ul preia linia respectivă.
- Lansează comanda în execuţie sub forma unui proces extern mediului Java.
- Captează ieşirea standard generată de execuţia comenzii şi o trimite clientului.
- Clientul afişează liniile pe consola lui

Sursa `ServerTelnet` este prezentată în programul 4.17.

```
import java.io.*;
import java.net.*;

public class ServerTelnet {

   public boolean telnet(BufferedReader in, PrintStream out)
                              throws Exception {
      String s = in.readLine();
      Process p = (Runtime.getRuntime()).exec(s);
      p.waitFor();
      InputStream pin = p.getInputStream();
      byte[] b = new byte[1000];
      int n = pin.read(b);
      out.print("Iesirea comenzii \""+s+"\" este: <###\r\n"+
                (new String(b,0,n))+"\r\n###>"+
                ", Lungime: "+n+" caractere, Cod de retur: "+
                 p.exitValue()+"\r\n");
      return p.exitValue()==0;
   } // ServerTelnet.telnet

   public static void main(String[] a) throws Exception {
      ServerTelnet st = new ServerTelnet();
      ServerSocket ss = new ServerSocket(5656);
      Socket s = ss.accept(); // S-a stabilit o conexiune
      BufferedReader in = new BufferedReader(
               new InputStreamReader(s.getInputStream()));
      PrintStream out = new PrintStream(s.getOutputStream());
      st.telnet(in, out);
      out.close();
      in.close();
      s.close();
      ss.close();
   } // ServerTelnet.main
} // ServerTelnet
```

Programul 4.17. `ServerTelnet.java`

Programul defineşte două metode. Metoda `telnet` primeşte ca şi parametri două obiecte `in` şi `out`, de tip `BufferedReader` şi `PrintStream` de care se foloseşte pentru a citi linii respectiv a scrie linii. Practic metoda nu ştie cu cine execută operaţii I/O aceste obiecte.

În cadrul metodei se citeşte o linie de la `in` care se presupune că e o comandă pentru SO-ul local. Cu ajutorul ei se creează un obiect de tip `Process` care lansează comanda în execuţie.

Se aşteaptă apoi terminarea execuţiei, după care se captează ieşirea standard a rulării comenzii, folosind în acest scop un obiect de tip `InputStream`.

În sfârşit, ieşirea captată se pune pe `out` împreună cu câteva informaţii suplimentare. Pentru a se vedea mai bine, liniile ieşirii standard generate de

execuția comenzii sunt puse între delimitatorii <### şi ###>. Informațiile suplimentare dau numărul de caractere al ieşirii standard şi codul de retur cu care s-a terminat execuția comenzii.

Metoda `main` a programului creează socket-ul de întâlnire (`ServerSocket`) şi aşteaptă prin `accept` contactarea de către client. Apoi, din obiectul socket întors de `accept` sunt create cele două obiecte I/O de schimb prin socket între client şi server. Apoi lansează metoda `telnet`, după care închide toate obiectele folosite.

Pentru exemplificare, am lansat sub Windows2000Pro comanda:

```
java  ServerTelnet
```

În acelaşi director am creat fişierul de comenzi `echo.bat`, prezentat în programul 4.18.

```
@echo off
echo Ii salut pe urmatorii:
:IAR
shift
if "%0" == "" goto :STOP
echo Salut %0
goto :IAR
:STOP
```

Programul 4.18. `echo.bat`

Apoi, de pe o maşină Linux am lansat un client `telnet`. Aspectul ferestrei client, urmare a execuției este prezentată în fig. 4.5.

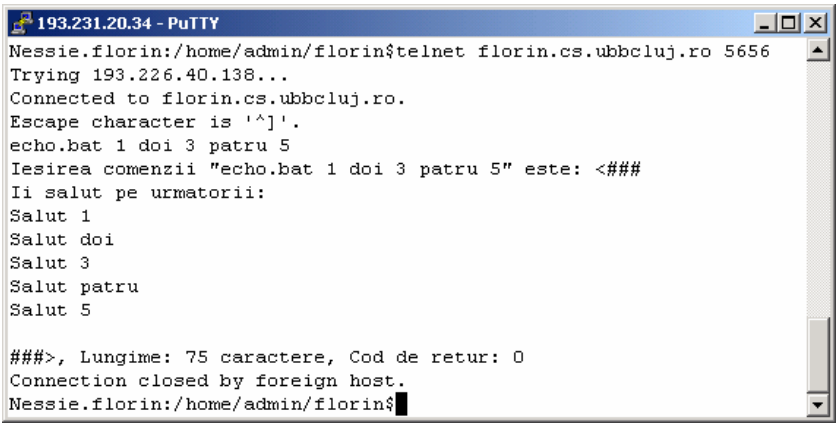

Figura 4.5. Fereastra client `telnet`

4.5.3.5. Miniserver Web

Comunicarea Web este o tehnologie clasică client/server. La *nivelul inferior*, Web comunică prin socket folosind *protocolul TCP*, iar mesajele schimbate sunt *linii de text*. Server-ul Web aşteaptă cereri de la clienţi fie la portul standard 80, fie la un alt port (de exemplu 8080 sau altul) fixat de administratorul server-ului Web.

La nivelul clientului se utilizează un *navigator* (*browser*) **Web**, cum ar fi **Mozilla, Internet Explorer, Netscape, Opera** etc. Prin intermediul browser-ului, un utilizator Internet lansează o *cerere* folosind o *specificare* URL.

Navigatorul se conectează la maşina cu server-ul Web invocat prin URL şi la portul solicitat. Din momentul conectării, între cei doi protagonişti începe un dialog bazat pe protocolul, de *nivel superior* indicat prin URL.

În cele ce urmează simplificăm lucrurile şi presupunem că se foloseşte numai protocolul **HTTP** (de fapt cel mai utilizat protocol), iar server-ul Web livrează navigatorului numai documente **HTML**. Pentru o documentare completă, invităm cititorul să consulte RFC1945, RFC2068 sau RFC2616 spre a vedea descrierea completă a protocolului HTTP.

În esenţă, prin HTTP comunicarea are loc în doi paşi. Mai întâi clientul iniţiază o conexiune şi trimite o *cerere* către server-ul Web, iar acesta îi întoarce un *răspuns.*

O *cerere HTTP* conţine trei componente:

- Prima linie indică *metoda de cerere* folosită, *URL*-ul resursei invocate şi *protocolul/versiunea* acestuia.
- Următoarele linii, până la prima linie goală (CRLF), conţin *antetul cererii*.
- Restul liniilor, dacă este cazul, conţin *corpul* propriu-zis al cererii.

De exemplu, o cerere ar putea fi formată din următoarele linii:

```
POST /cgi-bin/Cgi.cgi HTTP/1.1
Accept: text/plain, text/html, image/gif, image/x-xbitmap,
image/jpeg
Accept-Language: ro
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: florin:5555
Connection: Keep-Alive

Login=unUserOarecare&Password%3A=oParolaInClar
```

Un *răspuns HTTP* conţine trei componente:

- Prima linie indică *protocolul/versiunea* acestuia, *codul de răspuns* la cerere, *descrierea răspunsului*.

- Următoarele linii, până la prima linie goală, conțin *antetul răspunsului*.
- Restul liniilor, dacă este cazul, conțin *corpul* propriu-zis al răspunsului.

De exemplu, un răspuns posibil ar putea fi:

```
HTTP/1.1 200 OK
Server: Microsoft IIS/4.0
Date: Mon, 24 Nov 2003 11:07:32 GMT
Content-type text/html
Last Modified: Sun, 23 Nov 2003 11:07:32 GMT
Content-Length: 77

<HTML>
      <HEAD><TITLE>Prima pagina</TITLE></HEAD>
      <BODY>Salut</BODY>
</HTML>
```

Nu intrăm în detalii privind protocolul HTTP. Precizăm doar că tipurile de cereri folosite pot fi: GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE. Dintre aceste tipuri, ne vor interesa doar cererile de tip GET şi POST.

O cerere de tip GET este cea mai simplă şi probabil cea mai folosită metodă de cerere. Datele efective ale cererii sunt de volum relativ mic şi sunt furnizate în cadrul URL-ului de cerere. Securitatea datelor transmise este practic inexistentă, oricine poate să le citrească din URL. De exemplu, dacă în cererea de mai sus s-ar fi folosit metoda GET, atunci corpul cererii ar fi fost vid, iar prima linie ar fi fost:

```
GET /cgi-bin/Cgi.cgi?Login=
      unUserOarecare&Password%3A=oParolaInClar HTTP/1.1
```

O cerere de tip POST este folosită pentru a transmite datele efective ale cererii în corpul cererii. La acest tip de cerere volumul datelor nu este limitat şi există un grad oarecare de securizare a datelor transmise. În exemplul de cerere de mai sus am specificat o cerere POST.

În cele ce urmează vom defini un miniserver Web. Acesta va trata numai cereri de tip GET, iar cererile nu trebuie să conțină decât antet, nu şi corp. În cazul în care din linia GET a cererii lipseşte `URL`, se va considera implicit fişierul `index.html` aflat în directorul curent al miniserver-ului. Programul Java care joacă rol de server Web l-am numit `MiniHttpd` şi el se lansează:

```
java  MiniHttpd  port
```

Pentru a urmări mai uşor funcţionarea lui, din loc în loc am tipărit mesaje explicative. Unele dintre acestea sunt prefixate de numărul curent al cererii, altele conţin în plus informaţii calendaristice.

Pentru test, am depus în directorul curent al server-ului un singur fişier: `Salut.html`, cu următorul conţinut:

```
<html><head><title>Salut</title></head>
<body><h1>Salut</h1></body></html>
```

Pentru test, am specificat portul de ascultare 5555.

Prima cerere s-a făcut cu Mozilla de pe maşina Linux `nessie.cs.ubbcluj.ro` şi server-ul a fost indicat prin adresă Internet. În fig. 4.6 este ilustrată imaginea afişată de browser.

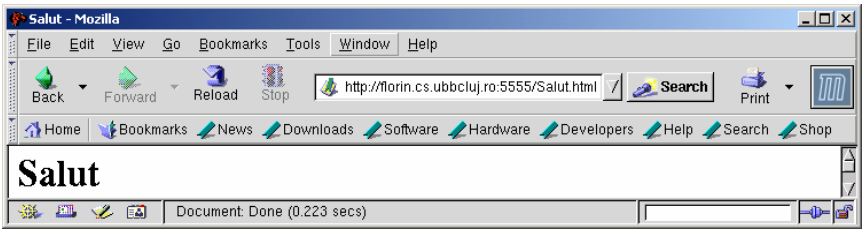

Figura 4.6. Primul apel `MiniHttpd`

Următoarea cerere, la fel cu prima, a fost făcută de pe aceeaşi maşină cu server-ul, care are Windows 2000 Pro. Navigatorul a fost tot Mozilla, iar adresa server-ului a fost indicată prin adresă IP. Imaginea afişată de browser este dată în fig. 4.7.

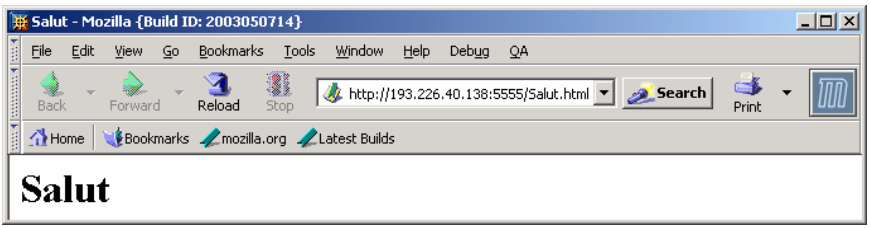

Figura 4.7. Al doilea apel `MiniHttpd`

Ultima cerere a fost făcută de pe maşina locală, cu Internet Explorer şi nu s-a specificat nici un fişier. Pe browser a apărut răspunsul ilustrat în fig. 4.8.

În fig. 4.9 sunt prezentate succesiunile de mesaje date de `MiniHttpd` la cele trei cereri de mai sus.

Figura 4.8. Al treilea apel `MiniHttpd`

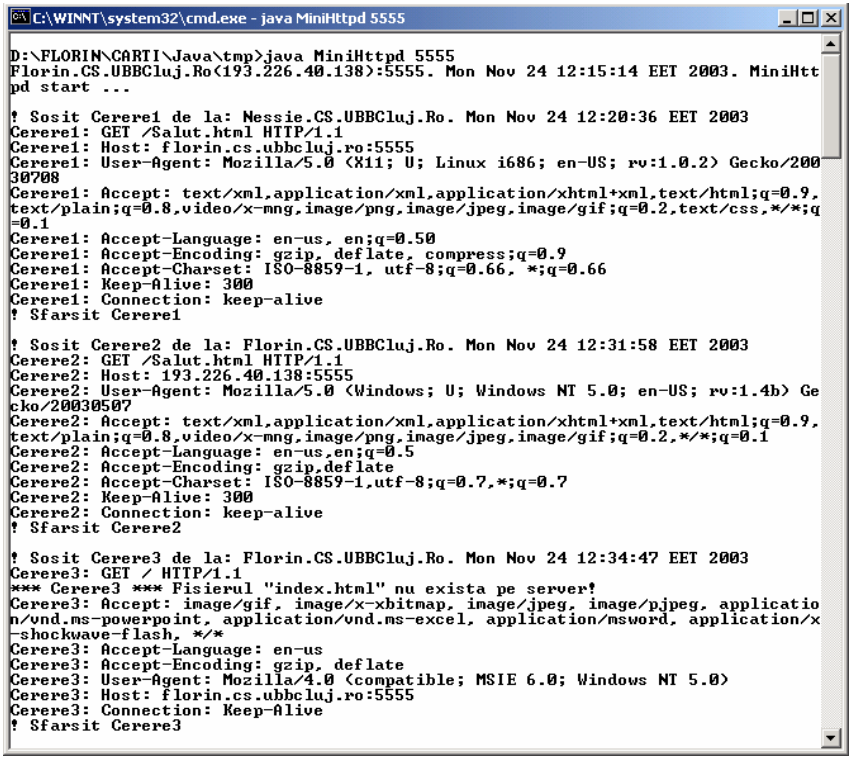

Figura 4.9. Fereastra `MiniHttpd`

Acum, după ce am văzut cum lucrează acest server, prezentăm şi textul lui sursă, în programul 4.19.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MiniHttpd {

   public static void main(String a[]) {
      if(a.length!=1) {
         System.out.println("Apel: java MiniHttpd port");
         System.exit(1);
      } // if
      new MiniHttpd(Integer.parseInt(a[0]));
   } // MiniHttpd.main

   public MiniHttpd(int port) {
      ServerSocket server;
      Socket socket;
      ThreadClient client;
      int nrCerere = 0;
      try {
         server = new ServerSocket(port);
         System.out.println(
            InetAddress.getLocalHost().getCanonicalHostName()
         +"("+InetAddress.getLocalHost().getHostAddress()+")"
         +":"+port+". "+new Date()+". MiniHttpd start ...\n");
         for ( ; ; ) {
            socket = server.accept(); // Asteapta conectarea de la un client
            nrCerere++;
            System.out.println("! Sosit Cerere"+nrCerere+
            " de la: "+
            socket.getInetAddress().getHostName()+". "+new Date());
            client = new ThreadClient(socket, nrCerere); // Creaza un
                                                         // thread pentru client
            client.start();
         } // for
      } // try
      catch(Exception e) {
         e.printStackTrace();
      } // catch
   } // MiniHttpd.run
} // MiniHttpd

class ThreadClient extends Thread {
   Socket socket;
   PrintStream out;
   BufferedReader in;
   int nrCerere;
```

```
public ThreadClient(Socket socket, int nrCerere) {
   this.socket = socket;
   this.nrCerere = nrCerere;
} // ThreadClient.ThreadClient

public void run() {
   String cerere;
   try {
      out = new PrintStream(socket.getOutputStream());
      in = new BufferedReader(
              new InputStreamReader(socket.getInputStream()));
      for ( ; ; ) {
         cerere = in.readLine();
         if (cerere==null)
            break;
         if (cerere.length()==0)
            break;
         System.out.println("Cerere"+nrCerere+": "+cerere);
         serveste(cerere);
      } // for cerere
      System.out.println("! Sfarsit Cerere"+nrCerere+"\n");
      in.close();
      out.close();
      socket.close();
   } // try
   catch (Exception e) {
      e.printStackTrace();
   } // catch
} // ThreadClient.run

void serveste(String cerere) {
   File file;
   DataInputStream dis;
   int n;
   byte[] b = null;
   if(cerere.length()<=0)
      return;
   // Se cauta linia de forma:  "GET /numeFisier HTTP1.1"
   if(!cerere.substring(0,3).equalsIgnoreCase("GET"))
      return;
   if (cerere.indexOf(" ") >= 0)
      cerere = cerere.substring(cerere.indexOf(" ")+1);
   if(cerere.charAt(0)=='/')
      cerere = cerere.substring(1);
   if (cerere.indexOf(" ") >= 0)
      cerere = cerere.substring(0, cerere.indexOf(" "));
   if(cerere.equals(""))
      cerere = "index.html";
   try {
      file = new File(cerere);
      if(file.exists()) {
         dis = new DataInputStream(new FileInputStream(file));
         n = (int) file.length(); // Accepta numai fisiere "scurte"
```

```
            b = new byte[n];

            dis.readFully(b);
            dis.close();
         } // if (then)
         else {
            System.out.println("*** Cerere"+nrCerere+
             " *** Fisierul \""+cerere+"\" nu exista pe server!");
            cerere = "<html><head><title>"+
              "Pagina de eroare</title></head>"+
              "<body><h1>HTTP/1.0 404  Not Found</h1><p>"+
              "<h3>Cerere"+nrCerere+": \""+cerere+"\":</h3>"+
              "<h4>nu exista pe server</h4></body></html>";
            b = cerere.getBytes();
         } // if (else)
         out.write(b);
         out.flush();
      } // try
      catch(Exception e) {
         e.printStackTrace();
      } // catch
   } // ThreadClient.serveste
} // ThreadClient
```

Programul 4.19. `MiniHttpd.java`

Programul este compus din două clase. Clasa `MiniHttpd` este clasa principală, conține metoda `main` şi constructorul. Metoda `main` preia numărul de port şi îl transmite constructorului `MiniHttpd`. Constructorul, în partea de inițializare creează obiectul de tip `ServerSocket` care ascultă la portul specificat. Pe lângă aceasta, la inițializare mai afişează un mesaj de start.

Urmează bucla principală a server-ului, aceea în care aşteaptă conexiuni de la clienți. La momentul unei conectări, prin intermediul metodei `accept`, se obține un obiect `Socket` pentru comunicarea cu clientul. După conectare se incrementează numărul curent al cererii şi se creează un obiect de tip `ThreadClient`. În fapt se creează un thread nou ce primeşte ca sarcină comunicarea cu clientul care s-a conectat. După creare, obiectul creat este lansat în execuție, iar bucla se reia în aşteptarea unei noi conexiuni.

Clasa `ThreadClient` are un constructor şi două metode: `run` şi `serveste`. Constructorul reține socket-ul prin care va comunica cu clientul şi numărul efectiv al cererii.

Metoda `run` a thread-ului `ThreadClient` descrie partea principală a comunicării cu un client. Mai întâi creează obiectele `in` şi `out` necesare schimburilor cu clientul. Apoi citeşte cererea de la client, linie cu linie. Afişează linia curentă a cererii, după care apelează metoda `serveste`, căreia îi trans-

mite linia de cerere. După terminarea servirii, închide obiectul, conexiunea socket şi tratarea clientului respectiv se termină.

Metoda `serveste` a thread-ului `ThreadClient` reţine din cerere doar linia cu tipul cererii şi URL-ul invocat. Dacă URL-ul este şirul vid, se fixează implicit fişierul `index.html`.

În cazul în care fişierul există, conţinutul lui este depus într-un şir de `bytes` (am ales această cale limitativă pentru a simplifica programul `MiniHttpd`). În cazul în care fişierul nu există, în şirul de `bytes` se introduce un scurt document HTML care anunţă eroarea.

Indiferent dacă fişierul există sau nu, metoda se încheie cu scrierea pe socket (spre client), a şirului de `bytes` care va ajunge la navigator.

4.6. Accesul la resurse Internet prin URL

4.6.1. Clasele URL şi URLConnection

4.6.1.1. Clasa URL

Clasa `URL`, acronim de la ***U**niform **R**esource **L**ocator*, permite instanţierea unor obiecte care identifică resurse accesibile pe Web. O astfel de resursă poate fi un fişier, un director, sau un obiect mai complex, cum ar fi o cerere adresată unei baze de date sau unui motor de căutare.

Reamintim că un URL are mai multe componente: protocolul folosit pentru accesarea resursei (http, ftp, etc), maşina pe care se găseşte resursa accesată, portul asociat serviciului care implementează protocolul folosit şi numele resursei respective. Numele resursei, include şi calea relativă a acesteia la rădăcina structurii de directoare a sistemului accesat.

Aceste componente, dintre care portul este opţional, sunt date ca parametri în constructorul folosit pentru iniţializarea unui obiect URL. Dacă nu este furnizat şi argumentul port, se consideră pentru acesta o valoare implicită (80 direct, 8080 prin *proxy*), care corespunde serviciului `http`.

Clasa `URL` conţine metode care pot fi folosite pentru obţinerea componentelor unui URL. Astfel `getFile` permite obţinerea numelui fişierului accesat, `getHost` numele maşinii, iar `getPort` numele portului la care s-a realizat conexiunea.

Datele sau obiectele referite printr-un URL pot fi aduse din Internet în trei moduri:

1. direct, prin intermediul metodei `getContent`, a clasei URL;
2. printr-un `InputStream`, obţinut folosind metoda `openStream` a clasei URL;

3. printr-o conexiune referită de un obiect `URLConnection`, obținut folosind metoda `openConnection` a clasei URL.

Clasa URL este definită astfel:

```
public final class URL extends Object {

      public URL(String protocol, String host,
            int port, String fisier) throws MalformedURLException;
      public URL(String protocol, String host,  String fisier)
            throws MalformedURLException;
      public URL(String specificare) throws MalformedURLException;
      public URL(URL context, String host)
                                    throws MalformedURLException;

      public String getProtocol();
      public String getHost();
      public int getPort();
      public String getFile();
      public String getRef();

      public String toExternalForm();

      public final Object getContent() throws IOException;
      public final InputStream openStream() throws IOException();
      public URLConnection openConnection() throws IOException;
}
```

4.6.1.2. Clasa URLConnection

Clasa abstractă `URLConnection` este superclasa tuturor claselor care intermediază comunicarea dintre o aplicație sau applet Java şi o resursă Internet, identificată printr-un URL. Astfel, instanțele acestei clase pot fi folosite pentru a primi sau a transmite date resursei contactate. În general, crearea unei conexiuni la un anumit URL se realizează în mai mulți paşi:

1. În prima etapă, se apelează metoda `openConnection` a clasei `URL` Această metodă returnează obiectul de conexiune de tipul `URLConnection`.
2. După apel se manipulează parametrii implicați în conexiunea la resursa de la distanță.
3. În a treia etapă, se apelează metoda `connect` a clasei `URLConnection`. Această metodă permite stabilirea unei conexiuni dintre aplicația curentă şi resursa referită prin URL.
4. După apelul `connect`, aplicația poate interacționa cu obiectul de la distanță referit prin URL.

În cadrul acestui scenariu de interacțiune, aplicația poate accesa câmpurile antetului conexiunii şi conținutul resursei Internet, cu ajutorul unor metode de tip `get`, dintre care amintim `getContent`, `getHeaderField`, `getInputStream` şi `getOutputStream`. Ultimele două metode permit deschiderea unui stream de intrare, respectiv a unui stream de ieşire, prin care aplicația curentă poate citi, respectiv transmite date obiectului de la distanță.

Printre parametri amintim două flag-uri care precizează statutul de intrare şi/sau ieşire. O conexiune poate fi folosită pentru intrare şi/sau pentru ieşire. Implicit flag-ul de intrare este `true`, iar flag-ul de ieşire este `false`.

Clasa `URLConnection` are multe câmpuri şi metode. Iată numai câteva dintre ele.

```
public final class URLConnection extends Object {

      public abstract void connect() throws IOException; // Deschide
                                                         // comunicatie
      public URL getURL(); // Intoarce obiectul URL al conexiunii
      public int getContentLength(); // Intoarce lungimea conținutului resursei
      public String getContentType(); // Intoarce valoarea campul Content-type
                                      // al resursei
      public long getDate(); // Data creare resursa, milisec. de la 01.01.1970 00:00
      public long getLastModified(); // Ca mai sus, data ultimei modificari

      // Urmatoarele doua metode intorc obiecte stream de acces la resursa
      public InputStream getInputStream() throws IOException();
      public getOutputStream();

      // Urmatoarele 4 metode manevreaza flagurile de intrare si de iesire
      public void setDoInput(boolean b);
      public boolean getDoInput();
      public void setDoOutput(boolean b);
      public boolean getDoOutput();

}
```

4.6.2. Exemplu de acces la o resursă Internet

Programul care urmează prezintă un exemplu de folosire a claselor `URL` (în mod indirect prin intermediul `u.getURL()`) şi `URLConnection`. Sunt afişate o serie de informații despre o resursă dată prin URL. Sursa este prezentată în programul 4.20.

```
import java.net.*;
import java.io.*;
import java.util.*;
```

```
public class URLInfo {
   public static void main(String a[]) throws Exception {
      // Creeaza un URL la o resursa, deschide o conexiune cu acest URL,
      URL url = new URL(a[0]);
      URLConnection u = url.openConnection();
      // Afiseaza adresa URL si alte informatii despre ea
      System.out.println(u.getURL().toExternalForm() + ":");
      System.out.println("Tip continut: " + u.getContentType());
      System.out.println("Lungime continut: "+
            u.getContentLength());
      System.out.println("Ultima modificare: "+
            new Date(u.getLastModified()));
      System.out.println("Flagurile I/O. Input: "+u.getDoInput()+
            " Output: "+u.getDoOutput());
      // Citeste si afiseaza primele 10 linii de la URL.
      System.out.println("Primele 10 linii:");
      BufferedReader in = new BufferedReader(
            new InputStreamReader(u.getInputStream()));
      for(int i = 0; i < 10; i++) {
         String linie = in.readLine();
         if (linie == null) break;
         System.out.println(linie);
      } // for
   } // URLInfo.main
} // URLInfo
```

Programul 4.20. `URLInfo.java`

Programul afişează, pentru URL-ul transmis ca argunent în linia de comandă, şase rânduri de informaţii despre URL-ul respectiv (URL-ul, tipul conţinutului, lungimea acestuia în octeţi, data şi ora ultimei lui modificări, flagurile de I/O) şi primele zece linii sursă ale acestui URL.

Fereastra de execuţie este ilustrată în fig. 4.10.

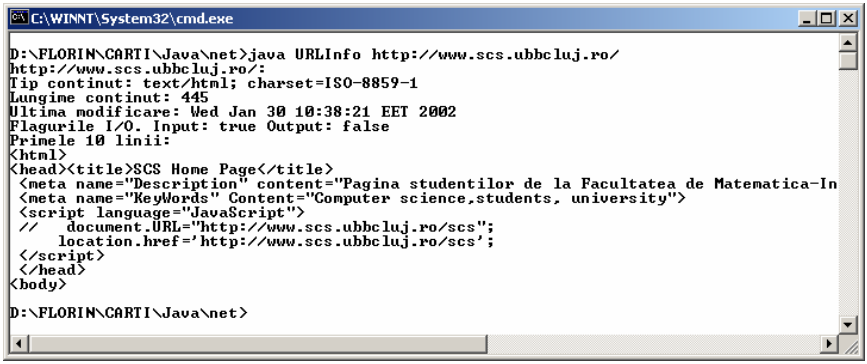

Figura 4.10. Fereastra execuţiei `URLInfo`

4.7. CGI şi comunicații prin URLConnection

4.7.1. Tehnologia CGI

Tehnologia **CGI**, acronim de la ***C**ommon **G**ateway **I**nterface* este una din cele mai vechi interfațe standard de comunicare dintre o aplicație program şi un client Web, prin intermediul unui server Web. Astfel, server-ul Web transmite cererea clientului, aplicației respective. Aceasta procesează cererea şi transmite rezultatul înapoi la server-ul Web, care la rândul său, forward-ează răspunsul la clientul care a lansat cererea.

Când utilizatorul solicită o pagină Web, server-ul, ca răspuns, îi trimite înapoi pagina respectivă. Dar dacă utilizatorul completează un formular HTML, server-ul transmite informațiile primite unui program CGI, care le procesează, după care trimite înapoi la server un mesaj de confirmare. Un document HTML este un fişier text static, care îşi păstrează o stare constantă. Pe de altă parte, un program CGI este executat în timp real şi generează în mod dinamic, un document HTML.

Un program CGI poate fi scris într-o diversitate de limbaje de programare, cele mai populare fiind: **C, C++,** script-uri **Perl**, script-uri **shell Unix**, **Visual Basic**, **AppleScript**. Pare curios, dar Java lipseşte din lista limbajelor de dezvoltare a programelor CGI. Motivul se va vedea în ultimele două capitole ale prezentei lucrări (**Servlet** şi **JSP**).

Un program CGI este lansat în execuție de către programul daemon al server-ului Web (`httpd`). În documentul invocator al CGI se specifică printr-un URL locul unde este plasat programul CGI. Locul (locurile) destinate plasării programelor CGI sunt stabilite de către administratorul server-ului Web şi tot el stabileşte drepturile de creare de programe CGI de către utilizatori [97, 98, 100, 2].

În momentul lansării în lucru a unui program CGI, server-ul Web stabileşte pentru acesta un context specific. În cadrul acestui context se fixează valorile unor variabile de mediu specifice CGI. Principalele variabile de mediu fixate la lansarea unui CGI sunt:

`SERVER_NAME`	Numele sau adresa IP a server-ului Web
`SERVER_SOFTWARE`	Software-ul server-ului Web (de exemplu IIS sau Apache)
`SERVER_PROTOCOL`	Protocolul de comunicare
`SERVER_PORT`	Portul de aşteptare
`REQUEST_METHOD`	Metoda de cerere (de exemplu GET sau POST)
`SCRIPT_NAME`	Calea şi numele CGI (de ex. `/cgi-bin/TestCgi.cgi`
`DOCUMENT_ROOT`	Rădăcina server-ului Web

QUERY_STRING	Şirul de cerere din URL (prin metoda GET, [86])
REMOTE-HOST	Numele maşinii client
REMOTE_ADDR	Adresa IP a maşinii client
REMOTE_USER	Numele utilizatorului client
CONTENT_TYPE	Tipul MIME al cererii (de exemplu text/html)
CONTENT_LENGTH	Lungimea (în octeţi) a datelor transmise în corpul cererii
HTTP_USER_AGENT	Numele browser-ului client
HTTP_REFFERER	URL-ul documentului accesat de client înainte de CGI

Indiferent de limbajul de elaborare a CGI, trebuie precizată modalitatea de primire a informaţiilor de către CGI de la formular şi modalitatea de răspuns.

Datele de intrare sunt primite de CGI sub forma unei linii de text. Programul trebuie să privească această linie ca:

- valoarea variabilei de mediu QUERY_STRING, dacă programul CGI este apelat prin metoda *Get,* [86, 85, 24];
- prima linie din fişierul de intrare standard, dacă programul CGI este apelat prin metoda *Post,* [86, 85, 24].

Datele de ieşire din CGI sunt tipărite de către CGI la ieşirea lui standard şi sunt tratate de server-ul Web în conformitate cu specificaţiile HTTP [85, 23]. De cele mai multe ori ieşirea standard este un document HTML (sau orice alt tip de document inteligibil în browser) care este trimis browser-ului (sau altui client Web) ca şi răspuns.

După cum se va vedea în exemplele care urmează, schimbul de date dintre CGI şi client poate fi orice obiect, cu condiţia să fie *serializabil*. Conexiunile care suportă schimburi mai speciale, şi care asigură un nivel mai mare de secret al comunicaţiei, sunt cele efectuate prin metoda *Post*.

4.7.2. Transformarea unui şir printr-un CGI

Aplicaţia distribuită este formată din trei nivele (aşa numită arhitectură *three-tier*):

1. Nivelul *client*, care furnizează, rând pe rând, către nivelul următor – server-ul Web -, una sau mai multe linii de text care urmează a fi transformate.
2. Nivelul *server Web*, aflat în general pe o altă maşină decât clientul. Acesta preia liniile furnizate de client, le transmite nivelului CGI,

acesta le va transforma, le va întoarce spre server-ul Web, care la rândul lui le va comunica transformate clientului.

3. Nivelul *CGI* care execută efectiv transformarea liniilor de text.

Pentru descrierea plicației, programatorul trebuie să implementeze numai clientul şi CGI. Drept server Web îl vom folosi pe cel de pe maşina `linux.scs.ubbcluj.ro`.

4.7.2.1. Programul CGI upcase

Construim un CGI care citeşte de la intrarea standard un şir de caractere. După ce a terminat de citit intrarea standard, întoarce acelaşi şir, în care toate literele mici sunt transformate în litere mari. Sursa C a acestui CGI este dată în programul 4.21.

```
#include <string.h>
#include <stdio.h>

   main () {
      char s[1000];
      int i;
      for(i=0; ;i++) {
         s[i] = getchar();
         if (s[i]==EOF)
            break;
      } // for
      s[i] = 0;
      for (i=0; i<strlen(s); i++)
         if((s[i]>='a')&&(s[i]<='z'))
            s[i]+='A'-'a';
      printf("Content-type: text/plain\n\n");
      printf ("%s\n\n",s);
   } // main
```

Programul 4.21. `upcase.c`

Acest program va fi compilat sub Unix, iar rezultatul compilării (fişierul executabil) rezultat va fi depus în fişierul: `~/public_html/cgi-bin/upcase.cgi` Administratorul server-ului Web a fixat acest loc pentru depunerea programelor CGI.

4.7.2.2. Un client Java standalone Stand2CGI

Programul care urmează este o aplicație Java standalone care apelează, prin `URLConnection`, acest CGI. Ea citeşte de la intrarea ei standard un şir

de linii şi le timite către CGI. Apoi citeşte liniile pe care i le întoarce CGI şi le depune pe ieşirea standard. Sursa acestei aplicații este dată în programul 4.22.

```
import java.net.*;
import java.io.*;

public class Stand2CGI {
   static final String NL = System.getProperty("line.separator");
   static public void main(String a[]) {
      URL url;
      URLConnection urlCon;
      DataOutputStream out;
      BufferedReader in;
      String s = "";
      byte b[];
      try {
         url = new URL(a[0]+"/cgi-bin/upcase.cgi");
         urlCon = url.openConnection(); // Deschide conexiunea
         urlCon.setDoOutput(true);        // permite si scrierea
         System.out.println("Conexiune la URL: "+
                            url.toExternalForm());
         in = new BufferedReader(new InputStreamReader(System.in));
         out = new DataOutputStream(urlCon.getOutputStream());
         // Citeste liniile de la intrarea standard si le trimite la CGI
         for ( ; ((s=in.readLine()) != null); ) {
            s += NL;
            b = s.getBytes();
            out.write(b);
         } // for
         out.close();
         // Creaza un obiect care va citi linii de la CGI
         in = new BufferedReader(
                new InputStreamReader(urlCon.getInputStream()));
         // Citeste liniile venite de la CGI si le afiseaza pe iesirea standard
         for ( ; ((s=in.readLine()) != null); ) {
            System.out.println(s);
         } // for
         in.close();
      } // try
      catch(Exception e) {
         System.out.println("Stand2CGI: "+e);
      } // catch
   } // Stand2CGI.main
} // Stand2CGI
```

Programul 4.22. **`Stand2CGI.java`**

Rezultatul execuției apare ca în imaginea din fig. 4.11.

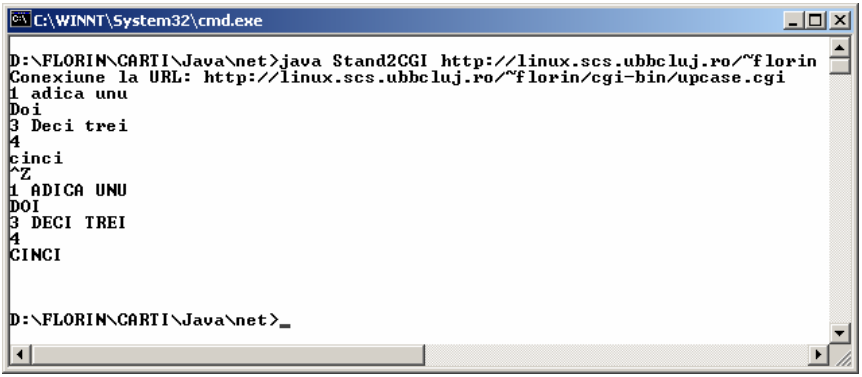

Figura 4.11. Rezultatul execuției clientului `Stand2CGI`

Singurul element care ni se pare demn de semnalat este faptul că la conexiune i se setează (pe `true`) flag-ul `DoOutput`.

4.7.2.3. Un client applet Applet2CGI

Documentul HTML prin care este apelat acest client va conține în interiorul lui liniile care vor fi transmise CGI-ului. Fiecare astfel de linie face obiectul unui tag <PARAM. Sursa acestui document este dată în programul 4.23.

```
<html>

   <head>
     <title>Applet ce apeleaza un CGI</title>
   </head>

<body>

   <APPLET code="Applet2CGI.class" width=500 height=300>
     <param name="URL" value="http://linux.scs.ubbcluj.ro/~florin">
     <param name="Linia1" value="Aceasta este linia 1">
     <param name="Linia2" value="Aceasta este linia 2">
     <param name="Linia3" value="Aceasta este linia 3">
     <param name="Linia4" value="Aceasta este linia 4">
     <param name="Linia5" value="Aceasta este linia 5">
     <param name="Linia6" value="Aceasta este linia 6">
     <param name="Linia7" value="Aceasta este linia 7">
     <param name="Linia8" value="Aceasta este linia 8">
     <param name="Linia9" value="Aceasta este linia 9">
     <param name="Linia10" value="Aceasta este linia 10">
     <param name="Linia11" value="Aceasta este linia 11">
     <param name="Linia12" value="Aceasta este linia 12">
```

```
   </APPLET>
</body>
</html>
```

Programul 4.23. `Applet2CGI.html`

Sursa applet-ului este dată în programul 4.24.

```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.*;

public class  Applet2CGI extends Applet {
   static final String NL = System.getProperty("line.separator");

   public void init() {
      URL url;
      URLConnection urlCon;
      DataOutputStream out;
      BufferedReader in;
      String s="", linie;
      byte b[];
      int i;
      TextArea raspuns = new TextArea();
      try {
         linie = getParameter("URL");
         url = new URL(linie+"/cgi-bin/upcase.cgi");
         s = "Conexiune la URL: "+url.toExternalForm()+NL;
         urlCon = url.openConnection(); // Deschide conexiunea
         urlCon.setDoOutput(true);         // Permite si scrierea in resursa
         out = new DataOutputStream(urlCon.getOutputStream());
         // Citeste liniile date ca parametri in HTML si le trimite la CGI
         for (i=1; ((linie = getParameter("Linia" + i))!= null);
                     i++) {
            linie += "\n";
            b = linie.getBytes();
            out.write(b);
         } // for
         // Creeaza un obiect care va citi linii de la CGI
         in = new BufferedReader(
               new InputStreamReader(urlCon.getInputStream()));
         // Citeste liniile de la CGI si le depune in s
         for (i=1; ((linie = in.readLine())!= null); i++) {
            s += linie+NL;
         } // for
         in.close();
         out.close();
         this.add(raspuns);    // Adauga obiectul la applet
         this.setVisible(true); // Fa applet-ul vizibil
         raspuns.setText(s);  // Depune liniile de raspuns
      } // try
```

```
        catch(Exception e) {
            s = "AppletCGI: "+e;
        } // catch
    } // Applet2CGI.init
} // Applet2CGI
```

Programul 4.24. `Applet2CGI.java`

Rezultatul execuției este dat în fig. 4.12.

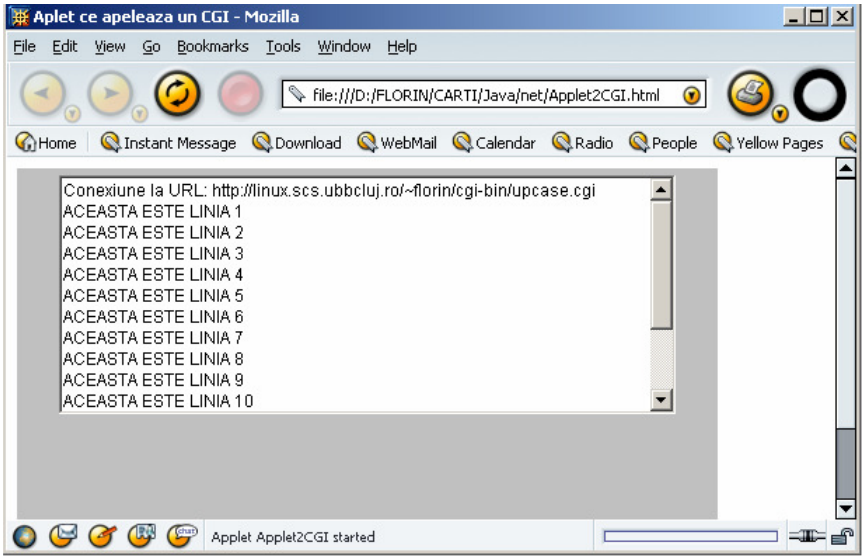

Figura 4.12. Rezultatul execuției applet-ului `Applet2CGI`

4.7.3. Construcția unui contor de pagini Web

Pentru construirea unui counter, folosim un program CGI şi un applet care să comunice cu acesta. CGI-ul întreține un fişier binar de 4 octeți care conține numărul ultimei consulări a unei anumite pagini. **Numele acestui fişier** este obținut din URL-ul CGI-ului, din care se păstrează doar literele şi cifrele numelui de CGI, iar toate literele mari sunt transformate în litere mici; la acest nume se mai adaugă sufixul `".count"`. De exemplu, dacă URL-ul este:

```
http://www.cs.ubbcluj.ro/~florin/cgi-bin/counterCGI.cgi
```

atunci fişierul întreținut de CGI are numele:

```
countercgicgi.count
```

4.7.3.1. Inițializarea fișierului contor

Programul 4.25 inițializează astfel de fișiere, preluând de la intrarea standard numele fișierului și valoarea inițială:

```
#include <stdio.h>
#include <fcntl.h>

   main (int c, char *a[]) {
      int i, f;
      if (c != 3) {
        printf(
           "Apel: counterInit numeFisier valoareInitialaContor");
        exit(1);
      } // if
      f = open(a[1], O_CREAT|O_WRONLY|O_TRUNC, 0666);
      i = atoi(a[2]);
      write(f, &i, sizeof(int));
      close(f);
   } // main
```

Programul 4.25. `initializareContor.c`

Pentru a putea fi apelat de către CGI, acest fișier trebuie să aibă cel puțin drepturile 0666!

4.7.3.2. Programul CGI

Acesta primește de la intrarea standard numele fișierului. Apoi îl deschide, citește valoarea curentă, o incrementează, o rescrie la loc și închide fișierul. După aceea trimite înapoi răspunsul către apelatorul CGI-ului, în conformitate cu protocolul HTTP pentru transmiterea de text simplu. Răspunsul constă dintr-o linie (de maximum 10 caractere) care conține numai cifrele zecimale ce reprezintă valoarea curentă a contorului.

Textul sursă al CGI este prezentat în programul 4.26.

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>

   extern int errno;

   main (int c, char *a[]) {
      int i=-1, f;
      char s[256];
      fgets(s, 256, stdin);    // Citeste numele fisierului
      f = open(s, O_RDWR);     // Deschide fisierul
```

```
      if (f > 0) {
         lockf(f, F_LOCK, 0);        // Blocheaza fisierul pentru incrementare
         read(f, &i, sizeof(int));   // Citeste numarul
         i++;                        // Incrementeaza
         lseek(f, 0, 0);             // Repozitioneaza la inceput
         write(f, &i, sizeof(int)); // Rescrie noul numar
         lockf(f, F_ULOCK, 0);       // Deblocheaza fisierul
         close(f);                   // Inchide fisierul
      } // if
      else
         perror("");
      printf("Content-type: text/plain\n\n%d\n", i);  // Trimite numarul
                                                      // la client
   } // main
```

Programul 4.26. `counterCGI.c`

Pentru cazul nostru, acest fişier executabil va fi fişierul:

```
~/public_html/cgi-bin/counterCGI.cgi
```

4.7.3.3. Applet-ul de apel al contorului

Pentru a se putea vizualiza contorul de acces al paginii, se utilizează un applet foarte simplu. Textul sursă al acestuia este dat în programul 4.27.

```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.*;
import java.util.*;

public class  AppletCounterCGI extends Applet {
   static final String nonLitereCifre =
         " ,./<>?;':\"[]{}`~!@#$%^&*()_-+=|\\";

   public void init() {
      URL url;
      URLConnection urlCon;
      BufferedReader in;
      DataOutputStream out;
      StringTokenizer st;
      String s = "";
      byte b[];
      Label raspuns;
      try {
         url = new URL(this.getCodeBase().toString()+
                       "cgi-bin/counterCGI.cgi");
         s = url.toString();
         s = s.substring(s.lastIndexOf("/"));
```

```
            st = new StringTokenizer(s, nonLitereCifre);
            s = "";
            while (st.hasMoreTokens()) {
               s += st.nextToken();
            } // while
            s = s.toLowerCase();
            s += ".count";
            urlCon = url.openConnection(); // Deschide conexiunea
            urlCon.setDoOutput(true);      // Permite scrierea
            out = new DataOutputStream(urlCon.getOutputStream());
            b = s.getBytes();
            out.write(b);
            System.out.println(s);
            in = new BufferedReader(    // Creeaza obiect ce va citi o linie din CGI
                     new InputStreamReader(urlCon.getInputStream()));
            s = in.readLine();          // Citeste linia si o depune in s
            raspuns = new Label(s, Label.CENTER);   // Depune linia
            this.add(raspuns);          // Adauga obiectul la applet
            this.setVisible(true);      // Fa applet-ul vizibil
            in.close();
            out.close();
         } // try

         catch(Exception e) {
            e.printStackTrace();
         } // catch
      } // AppletCounterCGI.init

} // AppletCounterCGI
```

Programul 4.27. `AppletCounterCGI.java`

În pagina HTML, invocarea acestui applet se face folosind secvența:

```
<APPLET code="AppletCounterCGI.class" width=50 height=30> </APPLET>
```

Numărul de apeluri va apare ca un obiect `Label` în centrul ferestrei rezervate de navigator pentru applet.

4.7.4. Istoricul consultării unei pagini Web

Vis-a-vis de o pagină Web, pare interesant de înregistrat o istorie a consultărilor ei. Deci, pe lângă numărarea acceselor, ne propunem să reținem, la fiecare acces al paginii, momentul accesului (după ceasul mașinii de pe care se accesează), momentul terminării consultării și, evident, adresa de la care s-a navigat.

Noi prezentăm un CGI şi un applet pentru aceasta şi lăsăm pe seama cititorului să reunească, eventual, contorizarea cu istoria. După cum se va vedea, momentele de start şi de terminare a accesului sunt obţinute de un applet.

Întreaga contorizare s-ar putea simplifica dacă se renunţă la momentele de timp ale maşinii client, eventual dacă în locul acestora s-ar folosi timpii de pe maşina server. În acest caz nu mai este necasar applet de invocare a CGI, iar adresa clientului se poate obţine folosind variabilele de mediu `REMOTE_HOST` şi `REMOTE_ADDR` ale procesului CGI.

4.7.4.1. Programul CGI

Programul CGI citeşte, prin metoda `post`, un şir de octeţi trimişi de applet-ul descărcat pe navigator. El scrie aceşti octeţi la sfârşitul unui fişier numit `history` şi mai scrie un caracter linie nouă `'\n'`. Sursa acestui CGI este dată în programul 4.28.

```
#include <stdio.h>

main () {
   int c;
   FILE *f;
   f = fopen("history", "a");
   for( ; ;) {
     c = getchar();
     if (c==EOF)
       break;
     putc(c, f);
   } // for
   c = '\n';
   putc(c, f);
   fclose(f);
   printf("Content-type: "+
     "text/plain\n\n<html><head></head><body></body></html>");
}// main
```

Programul 4.28. `history.c`

4.7.4.2. Applet-ul de acces

Applet-ul de acces rescrie metodele `init` şi `destroy`. În `init` obţine momentul de început şi adresa maşinii de pe care se navighează, iar în `destroy` obţine momentul de sfârşit şi trimite CGI-ului linia cu cele trei informaţii.

Sursa applet-ului este dată în programul 4.29.

```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.*;
import java.util.*;

public class  AppletHistory extends Applet {
   static final String NL = System.getProperty("line.separator");
   URL url;
   URLConnection urlCon;
   DataInputStream in;
   DataOutputStream out;
   String start="", navigator="", masina="";
   byte b[];
   public void init() {
      try {
         masina = getParameter("URL");
         start = ""+new Date();
         navigator = InetAddress.getLocalHost().toString();
      } // try
      catch(Exception e) {
         e.printStackTrace();
      } // catch
   } // AppletHistory.init

   public void destroy() {
      try {
         url = new URL(masina+"/cgi-bin/history.cgi");
         urlCon = url.openConnection(); // Deschide conexiunea
         urlCon.setDoOutput(true); // Permite si scrierea in resursa
         in = new DataInputStream(urlCon.getInputStream());
         out = new DataOutputStream(urlCon.getOutputStream());
         start = start+" "+(new Date())+" "+navigator;
         b = start.getBytes();
         out.write(b);
         out.close();
         in.close();
      } // try
      catch(Exception e) {
         e.printStackTrace();
      } // catch
   } // AppletHistory.destroy

} // AppletHistory
```

Programul 4.29. `AppletHistory.java`

De obicei, URL-ul la care se află CGI-ul şi fişierul `history` se află pe aceeaşi maşină cu pagina. Noi permitem ca CGI şi fişierul să poată fi, eventual pe o altă maşină. Din această cauză dat în codul HTML de citare a applet-ului

se specifică, printr-un tag <PARAM, URL-ul locului unde se află CGI-ul. Astfel, în documentul HTML, citarea applet-ului se face astfel:

```
<APPLET code="AppletHistory.class" width=10 height=10>
    <param name="URL" value="http://linux.scs.ubbcluj.ro/~florin">
</APPLET>
```