

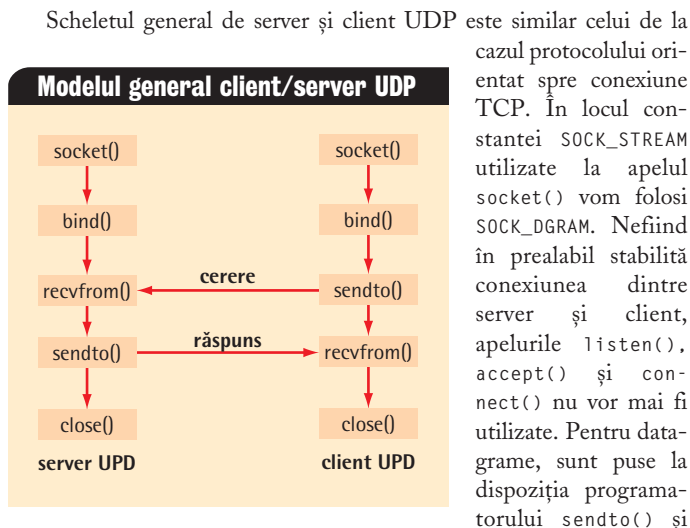
Aplicații Client/Server în C și JAVA – studiu comparativ (II)

Programarea în rețea (UDP și DNS)

– Ștefan Andrei, Sabin Corneliu Buraga

Socketuri pentru protocoale fără conexiune în C

A doua modalitate de transmisie a datelor în rețea este cea prin intermediul *datagramelor*, folosindu-se protocolul de transport fără conexiune UDP.



recvfrom(), primitive care pot fi utilizate pentru a trimite și, respectiv, a recepționa date de la un anumit client.

Modelul general al unui server/client UDP iterativ respectă ordinea următoarelor apeluri de sistem (vezi și figura):

- socket() va crea un socket care va trata conexiunile cu clienții;
- pregătirea structurilor de date (conținute în struct sockaddr_in) pentru a atașa socket-ul la portul folosit de aplicație;
- bind() atașează socket-ul la port;
- procesarea cererilor clientului – schimb de mesaje între server și client, prin intermediul primitivelor sendto() și recvfrom(); se pot utiliza, de asemenea, și primitivele generale send() și recv();
- close() închiderea socket-ului client.

Apelul recvfrom() are sintaxa generală:

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int recvfrom(int sockd, void *buf, size_t len, int flags,
             struct sockaddr *from, socklen_t *fromlen);
```

Apelul sendto() are următoarea formă:

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int sendto(int sockd, const void *msg, size_t len, int
           flags,
           const struct sockaddr *to, socklen_t tolen);
```

Drept exemplu, vom scrie un client UDP care se conectează la portul 13 (daytime) al unei mașini (server) pentru a citi timpul curent și a-l afișa la ieșirea standard.

O altă noutate care poate fi observată în sursa clientului este aceea că utilizatorul, pentru a se conecta la serverul daytime, nu va mai trebui să specifice adresa IP a mașinii pe care rulează acel server, ci adresa simbolică, în forma standard dată de DNS (*Domain Name Service*).

Acest lucru se realizează prin intermediul funcției gethostbyname() al cărei prototip este:

```
#include <netdb.h>
```

```
struct hostent *gethostbyname(const char *name);
```

Funcția va returna un pointer la tipul hostent având definiția:

```
struct hostent {
    char *h_name; /* numele oficial al gazdei */
    char **h_aliases; /* alias-uri ale gazdei */
    int h_addrtype; /* tipul de adresa al gazdei
                    (AF_INET) */
    int h_length; /* lungimea adresei */
    char **h_addr_list; /* lista de adrese IP */
}
/* prima adresa IP corespunzătoare gazdei */
#define h_addr h_addr_list[0]
```

În caz de eroare, se va returna NULL, iar variabila h_errno va conține codul numeric al erorii (errno nu se poate utiliza). Pentru a vedea descrierea erorii survenite, se poate folosi funcția perror() definită tot în fișierul antet netdb.h. O funcție înrudită cu gethostbyname() este funcția gethostbyaddr().

Posibile rezultate în urma rulării acestui program sunt arătate în figura „Rezultate posibile“.

Rezultate posibile



Listing 1. Client UDP pentru serviciul „daytime“

```

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

#define PORT 13 /* portul de conectare (daytime) */

main (int argc, char *argv[])
{
    int sd, i;
    struct sockaddr_in adr_dest; /* adresa serverului
    'daytime' */
    struct hostent *ip_addr; /* adresa gazdei */
    char s[70], st[70];

    /* verificam argumentele date in linia de comanda */
    if (argc != 2)
    {
        fprintf (stderr, "Sintaxa: %s <adr_server>\n",
            argv[0]);
        exit (1);
    }

    /* incercam sa gasim adresa IP */
    if ((ip_addr = gethostbyname (argv[1])) == NULL)
    {
        perror ("Eroare la rezolvarea adresei.");
        exit (1);
    }

    /* creare socket */
    if ((sd = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
    {
        perror ("Nu s-a putut crea socket-ul.");
        /* raportare erori */
        exit (1);
    }

    /* completare structura "sockaddr_in" */
    memset (&adr_dest, sizeof (adr_dest), 0);
    adr_dest.sin_family = AF_INET;
    adr_dest.sin_port = htons (PORT);
    /* adresa IP o luam din structura returnata de gethostby-
    name() */
    memcpy (&adr_dest.sin_addr.s_addr,
        ip_addr->h_addr, sizeof (ip_addr->h_addr));

    /* afisam adresele simbolice si IP a masinii la care ne vom
    conecta */
    printf ("Ne conectam la masina %s (%s) - portul %d...\n",
        ip_addr->h_name, inet_ntoa(adr_dest.sin_addr), PORT);
    /* initierea dialogului cu serverul */
    if (sendto (sd, s, 70, 0,
        (struct sockaddr *) &adr_dest, sizeof (adr_dest))
        < 0)
    {
        perror ("Eroare la sendto().");
        exit (1);
    }

    /* receptarea raspunsului */
    if ((i = recvfrom (sd, s, 70, 0,
        (struct sockaddr *) NULL, (int *) NULL)) < 0)
    {
        perror ("Eroare la recvfrom()");
        exit (1);
    }

    /* decupam timpul curent */
    st = strpbrk (s, " ") + 1;
    st = strpbrk (st + 1, " ") + 1;
    st = strpbrk (st + 1, " ");
    strtok (st, " ");
    printf ("Timpul curent: %s\n", st);

    /* am terminat */
    close (sd);
    exit (0);
}

```

Socketuri pentru protocoale fără conexiune. Varianta JAVA

În JAVA, socketurile pentru protocoale fără conexiune (în speță UDP) sunt implementate în clasele DatagramPacket și DatagramSocket.

Clasa DatagramPacket. Trimiterea unui pachet UDP implică crearea unui obiect DatagramPacket care constă din corpul mesajului și adresa destinație. Apoi acest obiect DatagramPacket poate fi pus în rețea în vederea trimiterii sale. Primirea unui pachet UDP implică crearea unui obiect DatagramPacket și apoi acceptarea unui pachet UDP din rețea. După primire, se poate extrage din obiectul DatagramPacket adresa sursă și conținutul mesajului.

Constructorii clasei DatagramPacket. Există doi constructori pentru datagrammele UDP. Primul constructor este folosit pentru primirea de pachete și necesită doar furnizarea unei memorii buffer, iar celălalt folosit pentru trimiterea de pachete necesită și specificarea adresei:

- DatagramPacket(byte buffer[], int lungime); Acesta este folosit pentru primirea de pachete. Parametrul buffer va memora conținutul pachetului care sosește (acesta trebuie să aibă deja alocată memorie). Parametru lungime este numărul maxim de octeți care ar trebui citiți în acest buffer. Orice dată în plus din pachet nu va fi memorată;
- DatagramPacket(byte buffer[], int lungime, InetAddress adresa, int port); Acesta este folosit pentru crearea unui pachet datagramă pentru transmisie. Corpul pachetului constă din primii lungime octeți ai șirului buffer. Pachetul va fi trimis către portul specificat și gazda specificată. Trebuie să existe un server UDP care ascultă la portul specificat pentru trimiterea pachetelor. Atenție! Numerele de port UDP și TCP sunt complet independente, deci putem avea și un server UDP și un server TCP care ascultă la același port.

Metodele clasei DatagramPacket. Dacă pachetul este primit, atunci adresele corespund gazdei sursă, iar dacă pachetul a fost creat pentru transmisie, atunci adresele corespund gazdei destinație. Conținutul și adresa unui obiect DatagramPacket pot fi interogate și modificate cu metodele:

- InetAddress getAddress(); returnează adresa IP a sursei pachetului (sau destinației);
- int getPort(); returnează numărul portului sursei pachetului (sau destinației). Astfel se poate determina numărul de port al serverului sursă direct;
- byte[] getData(); este folosită pentru extragerea datelor pachetului într-un șir de octeți. Acesta este același buffer specificat în constructor, deci va avea lungimea bufferului inițial și nu lungimea pachetului;
- int getLength(); este folosită pentru găsirea lungimii pachetului UDP, care poate să difere de lungimea bufferului;
- void setAddress(InetAddress adresa); (JDK 1.1) schimbă adresa IP a sursei pachetului la adresa specificată;
- void setPort(int port); (JDK 1.1) schimbă numărul portului sursei pachetului.
- void setData(byte[] buffer); (JDK 1.1) este folosită pentru schimbarea datelor pachetului în șirul buffer precizat;
- void setLength(int lungime); (JDK 1.1) este folosită pentru schimbarea lungimii pachetului DatagramPacket la valoarea specificată.

Clasa DatagramSocket. Această clasă se poate folosi atât pentru trimiteri, cât și pentru primiri de obiecte DatagramPacket. Ca și în cazul TCP, un obiect DatagramSocket trebuie să asculte la un număr de port particular între 1 și 65535 (porturile 1-1023 sunt în rezervate de obicei pentru aplicații sistem). Deoarece UDP nu este orientat pe conexiune, se va crea un singur obiect DatagramSocket pentru trimiterea pachetelor către diferite destinații și primirea pachetelor de la diferite surse. Nu există nici un control pentru a ști de la care gazde surse vin acele pachete.

Constructorii clasei DatagramSocket. Un obiect DatagramSocket va asculta la un port particular UDP pe mașina locală pachetele care sosesc. Se poate specifica un port pentru obiectul DatagramSocket sau se poate lăsa sistemul de operare să-l asigneze. De obicei, serverul va alege un port particular cu care să opereze și clienții vor permite unui port

aleator să fie asignat. Numărul de port al clientului va fi automat înscris în fiecare pachet trimis. Există și posibilitatea legării unui obiect `DatagramSocket` la o adresă locală particulară pe o mașină căreia îi sunt asignate mai multe adrese IP.

- `DatagramSocket()` throws `SocketException`; acest constructor crează un obiect `DatagramSocket` cu un număr de port ales aleator;
- `DatagramSocket(int port)` throws `SocketException`; acest constructor crează un obiect `DatagramSocket` care ascultă la un număr de port specificat;
- `DatagramSocket(int port, InetAddress locala)` throws `SocketException`; (JDK 1.1) acest constructor crează un obiect `DatagramSocket` care ascultă la un număr de port specificat și este legat la adresa locală specificată. Pentru mașinile căreia îi sunt asignate mai multe adrese IP, pachetele sunt trimise prin interfața rețelei corespunzătoare și deci par să vină de la adresa IP specificată. Pachetele care vin vor fi primite doar prin această adresă.

Metodele clasei `DatagramSocket`. Clasa `DatagramSocket` furnizează metode pentru trimiterea și primirea de obiecte `DatagramPacket` relative la închiderea socketului, determinarea informațiilor adresei locale și setarea timpului de primire.

- `void send(DatagramPacket pachet)` throws `IOException`; această metodă trimite *pachet* prin rețea. Dacă se trimit mereu pachete la o destinație inexistentă sau care nu ascultă, în cele din urmă se aruncă o excepție `IOException`;
- `void receive(DatagramPacket pachet)` throws `IOException`; această metodă primește un singur pachet UDP în obiectul *pachet* specificat. Pachetul poate fi apoi inspectat pentru determinarea adresei IP sursă, portului sursă și lungimii sale. Execuția metodei este blocată până când se primește cu succes un pachet sau se scurge timpul de așteptare;
- `InetAddress getLocalAddress();` (JDK 1.1) această metodă returnează adresa locală către care este legat acest `DatagramSocket`;
- `int getLocalPort();` această metodă returnează numărul de port unde ascultă `DatagramSocket`;
- `void close();` această metodă închide `DatagramSocket`.
- `void setSoTimeout(int timpDeAsteptare)` throws `SocketException`; (JDK 1.1) această metodă setează timpul de așteptare (în miliseunde) a unui socket la valoare specificată. Metoda `receive()` se va bloca doar pentru timpul de așteptare specificat pentru primirea unui pachet UDP, după care va arunca o excepție `InterruptedException`. Setarea timpului la 0 va dezactiva această proprietate astfel că operațiile socketului sunt blocate mereu;
- `int getSoTimeout()` throws `SocketException`; (JDK 1.1) această metodă returnează timpul de așteptare a socketului curent sau 0 dacă acesta nu este setat;
- `void setSendBufferSize(int lungime)` throws `SocketException`; (JDK 1.2) această metodă cere sistemului de operare să seteze lungimea bufferului de trimitere a socketului (`SO_SNDBUF`) la valoarea specificată. Pachetele UDP mai mari de această valoare nu pot fi trimise;
- `int getSendBufferSize()` throws `SocketException`; (JDK 1.2) această metodă returnează lungimea curentă a bufferului de trimitere a socketului;
- `void setReceiveBufferSize(int lungime)` throws `SocketException`; (JDK 1.2) această metodă cere sistemului de operare să seteze lungimea bufferului de primire a socketului (`SO_RCVBUF`) la valoarea specificată. Pachetele UDP mai mari de această valoare nu pot fi primite;
- `int getReceiveBufferSize()` throws `SocketException`; (JDK 1.2) această metodă returnează lungimea curentă a bufferului de primire a socketului;
- `void connect(InetAddress adresa, int port)` throws `SocketException`; (JDK 1.2) această metodă conectează acest socket la adresa și portul specificat la distanță. În general, această metodă se folosește

din motive de performanță și nu este cerută pentru operațiile uzuale UDP;

- `void disconnect();` (JDK 1.2) această metodă deconectează socketul conectat;
- `InetAddress getInetAddress();` (JDK 1.2) această metodă returnează obiectul `InetAddress` către care este conectat socketul, sau null dacă acesta nu este conectat;
- `int getPort();` (JDK 1.2) această metodă returnează portul către care este conectat socketul, sau -1 dacă acesta nu este conectat.

La crearea unui obiect `DatagramSocket`, se poate arunca o excepție `IOException` (sau din subclasa `SocketException`) la trimiterea sau primirea pachetelor dacă apar probleme.

Excepțiile de securitate (`SecurityManager`) restricționează accesul transmisiunilor și primirilor UDP. De exemplu, appleturile nesigure nu pot trimite și nici primi date de la alt server decât serverul inițial. Restricțiile de primire sunt rezolvate prin verificarea adresei sursă a fiecărui pachet primit. Dacă un pachet vine dintr-o sursă nevalidă, atunci acesta este aruncat și metoda `receive()` continuă să aștepte pachete valide.

Principiile unei aplicații client/server pentru un protocol fără conexiune

Comunicarea cu un protocol fără conexiune este mai simplu decât un protocol orientat pe conexiune, deoarece atât clientul, cât și serverul, folosesc obiecte `DatagramSocket`. Codul pentru programul la partea server are următoarele principii (etape):

- crearea unui obiect `DatagramSocket` asociat cu un număr de port specificat;
- crearea unui obiect `DatagramPacket`;
- cere obiectului `DatagramSocket` să pună următoarele date primite în obiectul `DatagramPacket`.

La partea client, ordinea este oarecum inversată:

- crearea unui obiect `DatagramSocket`;
- crearea unui obiect `DatagramPacket` asociat cu niște date, o adresă a rețelei destinație și un număr de port;
- cere obiectului `DatagramSocket` să trimită datele asociate în obiectul `DatagramPacket` către destinația asociată cu obiectul `DatagramSocket`.

Un exemplu complet de folosire a claselor `DatagramPacket` și `DatagramSocket`

Vom aplica șablonul de mai sus în vederea scrierii unei aplicații client/server care folosește protocolul UDP. Clientul va cere serverului timpul curent și serverul îl va furniza.

Clasa `ServerTimp`. Metoda `main()` începe prin crearea unui obiect `DatagramSocket` care utilizează portul 2001. Apoi creăm un obiect `DatagramPacket` care va conține date primite prin obiectul `DatagramSocket`. Când obiectului `DatagramSocket` i se cere să primească un pachet în obiectul `DatagramPacket`, este citit doar numărul de octeți specificați. În final, serverul intră într-o buclă infinită care primește cereri de la clienți folosind metoda `receive()` pentru obiectul `DatagramSocket` și apoi trimite răspunsuri.

Metoda `raspunde()` se ocupă de trimiterea răspunsurilor. Vom scrie timpul curent ca o valoare *long* într-un șir de octeți. Apoi pregătim trimiterea șirului de octeți prin crearea unui obiect `DatagramPacket` care încapsulează șirul, adresa și numărul de port al clientului care solicită timpul curent. În final, se trimit datele folosind metoda `send()` aplicată obiectului `DatagramSocket`.

Clasa `ClientTimp`. Metoda `main()` verifică mai întâi numărul corect de argumente (trebuie să fie un argument cu adresa IP). Apoi se creează un obiect `DatagramSocket` pentru comunicarea cu serverul. Urmează crearea unui obiect `DatagramPacket` care va conține cererea ce trebuie trimisă către server folosind metoda `send()` aplicată obiectului `DatagramSocket`. Apoi creăm încă un obiect `DatagramPacket` pentru primirea răspunsului de la server folosind metoda `receive()` aplicată obiectului

Listing 2. Codul claselor Java: ServerTimp și ClientTimp

```

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

public class ServerTimp
{
    static DatagramSocket socket;

    public static void main(String[] argv)
    {
        try
        {
            socket = new DatagramSocket(2001);
        }
        catch (SocketException e)
        {
            System.err.println("Nu putem crea socketul" +
                e.getMessage());
            System.exit(1);
        }
        DatagramPacket datagrama;
        datagrama = new DatagramPacket(new byte[1], 1);
        while (true)
        {
            try
            {
                socket.receive(datagrama);
                raspunde(datagrama);
            }
            catch (IOException e)
            {
                System.err.println("Nu putem primi si trimite
                    datagrama"
                    + e.getMessage());
            }
        }
    } // sfarsitul definitiei metodei main()

    static void raspunde(DatagramPacket cerere)
    {
        ByteArrayOutputStream baos;
        baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        // trimitem timpul curent in milisecunde de la server
        // catre client
        try
        {
            dos.writeLong(System.currentTimeMillis());
        }
        catch (IOException e)
        {
            System.err.println("Nu putem trimite timpul
                curent"
                + e.getMessage());
        }
        DatagramPacket raspuns;
        byte[] data = baos.toByteArray();
        raspuns = new DatagramPacket(data, data.length,
            cerere.getAddress(), cerere.getPort());
        try
        {
            socket.send(raspuns);
        }
        catch (IOException e)
        {
            System.err.println("Nu putem trimite datagrama"
                + e.getMessage());
        }
    } // sfarsitul definitiei metodei raspunde
} // sfarsitul definitiei clasei ServerTimp

// incepe definitia clasei ClientTimp - fisier ClientTimp.java
import java.net.*;
import java.io.*;
import java.util.*;

public class ClientTimp
{
    public static void main(String argv[])
    {
        if (argv.length != 1)
        {
            System.out.println("Dati comanda: ClientTimp
                numeServer");
            System.exit(1);
        }
        DatagramSocket socket = null;
        try
        {
            socket = new DatagramSocket();
        }
        catch (SocketException e)
        {
            System.err.println("Nu putem crea socketul"
                + e.getMessage());
            System.exit(1);
        }
        long timp = 0;
        try
        {
            byte[] buffer = new byte[1];
            socket.send(new DatagramPacket(buffer, 1,
                InetAddress.getByAddress(argv[0]), 2001));
            DatagramPacket raspuns = new DatagramPacket(new
                byte[8], 8);
            socket.receive(raspuns);
            ByteArrayInputStream baos;
            baos = new
                ByteArrayInputStream(raspuns.getData());
            DataInputStream dis = new DataInputStream(baos);
            timp = dis.readLong();
        }
        catch (IOException e)
        {
            System.err.println("Nu putem trimite si primi
                datagrama"
                + e.getMessage());
        }
        System.out.println("Timpul primit de la server
            este "
            + new Date(timp));
        socket.close();
    } // sfarsitul definitiei metodei main()
} // sfarsitul definitiei clasei ClientTimp

```

DatagramSocket. În final, clientul afișează timpul curent și închide socketul (indiferent dacă se aruncă o excepție sau nu!).

Clasa *InetAddress*. Datele pot fi trimise prin rețea, folosind adresa IP specifică mașinii de destinație. Clasa *InetAddress* furnizează acces abstract la adresele IP. Avantajul folosirii acestei clase pentru reprezentarea unei adrese IP într-un întreg pe 32 biți, este că aplicațiile vor fi transparent portabile la IPv6 (IP versiunea 6), care furnizează adrese pe 128 de biți.

Nu există constructori pentru această clasă. Instanțele trebuie create folosind metodele statice *getByName()*, *getLocalHost()* și *getAllByName()*. Acestea au sintaxa:

- *InetAddress getLocalHost()* throws *UnknownHostException*; Aceasta metodă returnează un obiect *InetAddress* corespunzător mașinii locale. În anumite cazuri (de exemplu, pentru un parafoc), aceasta va fi adresa „loopback“ 127.0.0.1
- *InetAddress getByAddress(String host)* throws *UnknownHostException*; Aceasta metodă returnează un obiect *InetAddress* pentru host-ul specificat. Host-ul poate fi specificat prin nume (de exemplu, proxy 0.att.com) sau prin adresa IP (de exemplu, 127.0.0.1). Nu

există alt mecanism client pentru crearea unui obiect *InetAddress* pentru o gazdă la distanță;

- *InetAddress [] getAllByName (String host)* throws *UnknownHostException*; Această metodă returnează un vector de obiecte *InetAddress* corespunzător fiecărei adrese IP cunoscute pentru host-ul specificat. De obicei, site-urile Web de trafic înalt au mai multe adrese IP pentru un singur nume de host.

Clasa *InetAddress* are următoarele metode instanță (adică asociate unui obiect al clasei, și nu clasei):

- *byte [] getAddress()*; Metoda întoarce un vector de octeți corespunzător adresei IP reprezentate de obiectul *InetAddress*. Vectorul este în ordinea octeților adresei rețelei, adică octetul semnificativ primul. Sub IPv4, acest vector are doar 4 octeți, sub IPv6 vor fi 16 octeți;
- *String getHostName()*; Metoda returnează numele host-ului reprezentat de obiectul



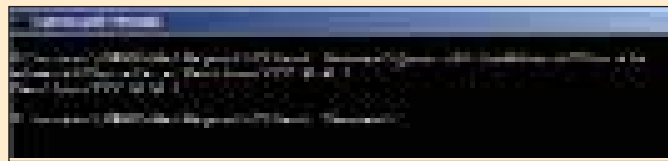
Exemplul 1. Un program JAVA care afișează adresa IP locală

Fișierul „obțineAdresaIPLocala.java” este:

```
import java.net.*;
public class obțineAdresaIPLocala
{
    public static void main(String args[])
    {
        InetAddress adresaIPLocala = null;
        try
        {
            adresaIPLocala = InetAddress.getLocalHost();
            System.out.println("adresaIPLocala = " +
                adresaIPLocala);
        }
        catch (UnknownHostException e)
        {
            System.out.println("Eroare: Nu putem gasi gazda "
                + e.toString());
        }
    }
}
```

O posibilă execuție a acestui program JAVA poate fi obținută prin comanda:

Java obțineAdresaIPLocala



InetAddress. Dacă numele hostului nu este cunoscut, atunci va încerca să-l caute folosind DNS sau folosind IP. Dacă această căutare eșuează, se returnează adresa IP în forma numerică;

- `String getHostAddress()`; Metoda returnează adresa IP a gazdei reprezentată prin obiectul `InetAddress` ca un `String` (de exemplu `127.0.0.1`);
- `boolean isMulticastAddress()`; Metoda returnează `true` dacă obiectul `InetAddress` reprezintă o adresă IP multicast (din clasa D); de fapt, primul octet trebuie să fie între 224 și 239.

Există două excepții noi care pot fi aruncate de cele trei metode statice.

- `UnknownHostException` - subclasă a clasei `IOException` și indică faptul că host-ul căutat nu a fost identificat cu succes;
- `SecurityException` - poate fi aruncată dacă `SecurityManager` nu permite o operație specifică. Un applet (de neîncredere) poate doar să construiască un obiect `InetAddress` pentru numele gazdei a serverului Web de unde provine codul lui.

Concluzii

Cum era de așteptat, JAVA încapsulează detaliile de implementare a protocolului UDP în clasele `DatagramPacket` și `DatagramSocket`. De asemenea, această facilitate este valabilă și pentru serviciul DNS. În C, acest lucru se realizează cu ajutorul primitivelor sistem implementate în nucleul sistemului de operare (UDP) și al funcțiilor de bibliotecă și structurilor de date oferite programatorilor (DNS).

Propunem cititorilor interesați, rezolvarea și implementarea următoarelor exerciții:

1. Scrieți programe C și JAVA care să specifice clasa adresei IP (din cele cinci existente: A, B, C, D, E) a unui host (la distanță) folosind numele de domeniu (sau numele unui server) ca dată de intrare. De exemplu, clasa C are valoarea primului octet între 193 și 224.

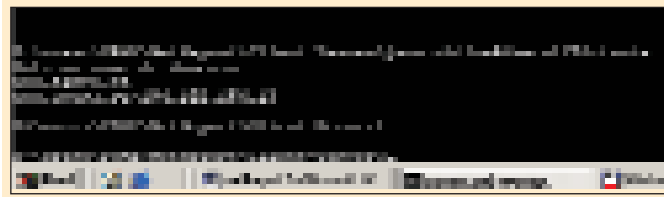
Exemplul 2. Un program JAVA care afișează adresa IP a unei mașini la distanță plecând de la numele de domeniu.

Fișierul „obțineAdresaIPDistanța.java” este:

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

public class obțineAdresaIPDistanța
{
    public static void main(String args[])
    {
        BufferedReader tastatura;
        tastatura = new BufferedReader(new
            InputStreamReader(System.in),1);
        System.out.println("Dati un nume de domeniu");
        String linie ="";
        // citim numele de domeniu reprezentat ca un String
        try
        {
            System.out.flush();
            linie = tastatura.readLine();
            tastatura.close();
        }
        catch (IOException e)
        {
            System.out.println("Exceptie in timpul citirii
                de la tastatura"+ e.toString());
            System.exit(2);
        }
        InetAddress adresaIP = null;
        try
        {
            adresaIP = InetAddress.getByLine(linie);
            System.out.println(adresaIP);
        }
        catch (UnknownHostException e)
        {
            System.out.println("Eroare: Nu putem gasi gazda "
                + e.toString());
        }
    } // de la public static void main(String args[])
} // de la public class obțineAdresaIPDistanța
```

O posibilă execuție a acestui program este:



2. Să se realizeze în C și JAVA un server și un client UDP care să implementeze un serviciu de conversație similar cu talk.

Autorii sunt cadre didactice la Facultatea de Informatică, Universitatea „Al.I.Cuza” din Iași și pot fi contactați prin email la adresele stefan@infoiasi.ro și busaco@infoiasi.ro, respectiv. ■ 98

Bibliografie

- [1] Budd, T.: Understanding Object-Oriented Programming with JAVA. Addison-Wesley. 2000
- [2] Buraga, S. C., Ciobanu, G. - Atelier de programare în rețele de calculatoare. Editura Polirom, Iași, 2001
- [3] Hughes, M., Shoffner, M., Hamner, D., Bellur, U.: JAVA. Network Programming. Manning. 1999
- [4] Grand, M., Knudsen, J.: JAVA. Fundamental Classes Reference. O'Reilly. 1997