* **Un server capabil să comunice cu mai mulţi clienţi**

Ideea de bază este de a porni câte un fir de executare pentru fiecare client. Schema pe care o vom urma este descrisă în continuare.

ServerSocket ss = new ServerSocket(port); Socket cs;

while(true) {

cs = ss.accept();

new Conexiune(cs); // crearea unui fir anonim

}

class Conexiune extends Thread {

Socket cs;

Conexiune(Socket cs) {

this.cs = cs; start();

}

public void run() {

*crearea de fluxuri şi schimbul de informaţii cu clientul*

}

}

Se observă că firele create sunt anonime; de aceea este necesar să invocăm start() din constructorul clasei Conexiune.

Schema de mai sus poate fi reprezentată astfel:

ss

cs1

cs2

is1

os1

Client 1

Client 2

is2

os2

Clienţi

Server

Fire

Firele se execută conform modelului aleator, ceea ce se poate formula aici astfel*: serverul trece în mod aleator de la un fir/client la altul, pentru intervale aleatoare de timp*. În particular, niciun client nu poate bloca serverul.

**Exemplu.** *Serverul afişează texte primite de la clienţi.*

Drept acţiune a clienţilor am ales ca, după conectarea lor la server, să transmită un text ce urmează a fi afişat de server. Pentru simplificare, în loc de text sunt transmise o cifră cifra şi un număr natural n cu intenţia ca serverul să afişeze de n ori cifra respectivă.

// Unitatea de compilare Client\_n.java

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

class Client\_n {

public static void main(String[] args) throws Exception {

Scanner sc = new Scanner(System.in);

System.out.print("Adresa serverului si portul : ");

Socket cs = new Socket( sc.next(), sc.nextInt() );

OutputStream os = cs.getOutputStream();

int c = (int) (10 \* Math.random());

// cere serverului sa afiseze de 200 de ori cifra c

os.write(c); os.write(200); os.close();

}

}

În esenţă serverul porneşte câte un fir de executare pentru fiecare client care se conectează la el. Mai precis:

* serverul crează un socket de server ss, care în mod repetat aşteaptă ca un client să se conecteze cu succes; când acest lucru se realizează, este obţinut un socket de client cs şi este pornit un fir de executare de tipul clasei Conexiune, fir căruia îi este comunicată referinţa la cs şi identitatea clientului (al câtelea este el conectat la server); apoi serverul reia aşteptarea conexiunii de la alt client;
* să observăm că firul nou creat este anonim; ca urmare acţiunea care trebuie să o întreprindă cade în sarcina constructorului. Concret, constructorul crează fluxurile ataşate lui cs şi invocă, prin start(), metoda run a clasei Conexiune; acţiunea care trebuie întreprinsă este inclusă în această metodă. În acest exemplu acţiunea constă în tipărirea de n de ori, la intervale variabile de timp, a cifrei cifra (ambele citite din fluxul de intrare).

// Unitatea de compilare Server\_n.java

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

class Server\_n {

static int i=0;

public static void main(String[] arg) throws IOException {

ServerSocket ss = null; Socket cs = null;

Scanner sc = new Scanner(System.in);

System.out.print("Portul : ");

ss = new ServerSocket( sc.nextInt() );

System.out.println("Serverul a pornit");

while (true) {

cs = ss.accept();

System.out.println("\nClient nou. ");

new Conexiune(cs,++i);

}

}

}

class Conexiune extends Thread {

int identitate; Socket cs = null; InputStream is = null;

public Conexiune(Socket client, int i)

throws IOException {

cs = client; identitate = i; is = cs.getInputStream();

start();

}

public void run() {

try {

int cifra = is.read(); int n = is.read();

System.out.println("Clientul " + identitate +

" a transmis:");

for (int k=0; k<n; k++) {

sleep((int) (50\*Math.random()) );

System.out.print(" " + cifra);

}

}

catch(Exception e) { }

}

}

*Observaţie*. În exemplul de mai sus firele corespunzătoare clienţilor folosesc o resursă comună, ceea ce conduce la probleme de concurenţă. Într-adevăr, chiar pentru doi clienţi, cele câte 200 de tipăriri de cifre apar intercalate. Eventuala asigurare a excluderii reciproce, sau rezolvarea altor probleme de concurenţă, cade în sarcina programatorului !

Prezentăm în continuare, fără comentarii, o modalitate de realizare a excluderii reciproce pentru exemplul anterior.

// Unitatea de compilare Server\_nn.java

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

class Server\_nn {

static int i=0;

public static void main(String[] arg) throws IOException {

ServerSocket ss = null; Socket cs = null;

Scanner sc = new Scanner(System.in);

System.out.print("Portul : ");

ss = new ServerSocket( sc.nextInt() );

System.out.println("Serverul a pornit");

while (true) {

cs = ss.accept();

System.out.println("\nClient nou. ");

new Conexiune(cs,++i);

}

}

}

class Conexiune extends Thread {

int identitate; Socket cs = null; InputStream is = null;

static C Ob = new C();

public Conexiune(Socket client, int i)

throws IOException {

cs = client; identitate = i; is = cs.getInputStream();

start();

}

public void run() {

try {

int c = is.read(); int n = is.read();

System.out.println("Clientul " + identitate +

" a transmis:");

Ob.scrie(c,n);

}

catch(Exception e) { }

}

}

class C {

synchronized void scrie(int c,int n) throws Exception {

for (int k=0; k<n; k++) {

Thread.sleep((int) (20\*Math.random()) );

System.out.print(" " + c);

}

}

}

* **Transmiterea fişierelor de la server la client (download)**

*Dorim să descărcăm un fişier aflat pe un server.*

Începem cu descrierea clientului/clienţilor.

Adresa serverului, portul şi numele fişierului căutat vor fi introduse de la intrarea standard. În fluxul de ieşire asociat socket-ului de client este înscris numele fişierului pe care dorim să-l descărcăm (download-ăm). Din fluxul de intrare asociat socket-ului client se citesc pe rând:

* un prim octet c, a cărui valoare este 1 dacă fişierul este găsit pe server, respectiv 0 în caz contrar;
* în cazul c=1, sunt citiţi pe rând octeţii trimişi de server; aceşti octeţi sunt înscrişi în fişierul local şi afişaţi la ieşirea standard. Fişierul recepţionat va fi inclus sub acelaşi nume în directorul curent.

// Unitatea de compilare Down\_C.java

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

class Down\_C {

public static void main(String[] sir) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.print("Adresa server si port : ");

Socket cs = null;

try {

cs = new Socket( sc.next(), sc.nextInt() );

}

catch(Exception e) {

System.out.println("Conectare esuata");

System.exit(0);

}

InputStream is = cs.getInputStream();

DataOutputStream dos =

new DataOutputStream(cs.getOutputStream());

System.out.print("Fisier cerut: ");

String fisier = sc.next(); dos.writeUTF(fisier);

int c = is.read();

if( c==0 ) {

System.out.println("Fisier inexistent");

System.exit(0);

};

OutputStream fos = new FileOutputStream(fisier);

while( (c=is.read()) != -1) {

System.out.print("" + (char) c); fos.write(c);

}

System.out.println("\nFisier primit!");

fos.close(); is.close(); dos.close(); cs.close();

}

}

Serverul va lansa câte un fir de executare pentru fiecare client ce se conectează la el. Va fi folosit acelaşi port pentru toţi clienţii, deoarece sunt oferite aceleaşi servicii. Fiecare fir de executare primeşte în fluxul de intrare asociat socket-ului cs numele fişierului dorit şi transmite clientului următoarele:

* un prim octet c a cărui valoare este 1 sau 0 după cum fişierul cerut este găsit sau nu pe server;
* în cazul c=1 este transmis clientului (prin înscriere octet cu octet în fluxul de ieşire asociat socket-ului cs) conţinutul fişierului cerut.

// Unitatea de compilare Down\_S.java

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

class Down\_S {

public static void main(String[] arg) throws IOException {

Scanner sc = new Scanner(System.in); Socket cs = null;

System.out.print("Port : ");

ServerSocket ss = new ServerSocket( sc.nextInt() );

System.out.println("Serverul a pornit");

while (true) {

cs = ss.accept();

System.out.print("Client nou. Fisier cerut: ");

new Conexiune(cs);

}

}

}

class Conexiune extends Thread {

Socket cs = null;

OutputStream os = null; DataInputStream dis = null;

public Conexiune(Socket client) throws IOException {

cs = client;

dis = new DataInputStream(cs.getInputStream());

os = cs.getOutputStream();

start();

}

public void run() {

String mesaj = null, fisier = null;

try {

fisier = dis.readUTF(); System.out.println(fisier);

}

catch(IOException e) { }

int c=1; FileInputStream f = null;

try { f = new FileInputStream(fisier); }

catch(FileNotFoundException e) { c = 0; }

try {

os.write(c);

if( c==1 ) {

while( (c = f.read()) != -1) os.write(c);

f.close(); System.out.println("Fisier transmis!");

}

else { System.out.println("Fisier inexistent!"); }

cs.close(); dis.close(); os.close();

}

catch(IOException e){ }

}

}

*Observaţie*. Să presupunem că atât serverul cât şi clienţii se află pe maşina gazdă, acţionând în câte o fereastră de comandă. Modelul aleator este pus în evidenţă de scrierea în fiecare fereastră asociată unui client a octeţilor primiţi de la server: scrierea comută aleator de la o fereastră la fereastră!

* **Transmiterea fişierelor de la client la server (upload)**

*Dorim să trimitem serverului un fişier local (aflat pe client).*

Începem cu descrierea clientului/clienţilor.

Numele fişierului ce urmează a fi trimis serverului este introdus de la intrarea standard. Dacă fişierul cu numele specificat nu este regăsit, programul se termină.

Numele fişierului poate include şi calea până la locaţia sa; va fi transmis însă numele propriu-zis, adică ce urmează după ultimul caracter '\'.

În continuare sunt citite de la intrarea standard adresa serverului şi portul şi este creat socket-ul de client.

În fluxul de ieşire asociat socket-ului sunt transmişi pe rând către server octeţii citiţi din fişierul respectiv.

// Unitatea de compilare Up\_C.java

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Up\_C {

public static void main(String[] sir) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.print("Fisier de transmis: ");

String fisier = sc.next();

InputStream fis = null;

try { fis = new FileInputStream(fisier); }

catch (FileNotFoundException e) {

System.out.println("Fisier inexistent");

System.exit(0);

}

Socket cs = null;

System.out.print("Adresa serverului si portul : ");

try {

cs = new Socket( sc.next(), sc.nextInt() );

}

catch(Exception e) {

System.out.println("Conectare esuata");

System.exit(0);

}

int p = fisier.lastIndexOf("\\");

fisier = fisier.substring(p+1,fisier.length());

DataOutputStream dos =

new DataOutputStream(cs.getOutputStream());

dos.writeUTF(fisier);

int c;

while( (c=fis.read()) != -1) dos.write(c);

fis.close(); System.out.println("Fisier transmis!");

dos.close(); cs.close();

}

}

Trecem la prezentarea serverului capabil să primească fişiere transmise de către clienţi, pe care le salvează sub acelaşi nume în directorul curent (din care este pornit). Va fi folosit acelaşi port pentru toţi clienţii. Fiecare fir de executare asociat unui client primeşte numele fişierului şi apoi (prin citirea unui octet din fluxul de intrare asociat socket-ului) conţinutul fişerului.

// Unitatea de compilare Up\_S

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Up\_S {

public static void main(String[] arg) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.print("Portul : ");

ServerSocket ss = new ServerSocket( sc.nextInt() );

System.out.println("Serverul a pornit");

Socket cs = null;

while(true) {

cs = ss.accept();

System.out.print("Client nou. ");

new Conexiune(cs);

}

}

}

class Conexiune extends Thread {

Socket cs;

DataInputStream dis;

public Conexiune(Socket client) throws IOException {

cs = client;

dis = new DataInputStream(cs.getInputStream());

start();

}

public void run() {

String mesaj = null, fisier = null;

try {

fisier = dis.readUTF();

System.out.println("Se primeste fisierul : "

+ fisier);

FileOutputStream fos = new FileOutputStream(fisier);

int c;

while( (c=dis.read()) != -1) fos.write(c);

fos.close(); System.out.println("Fisier primit !");

cs.close(); dis.close();

}

catch(IOException e) { }

}

}