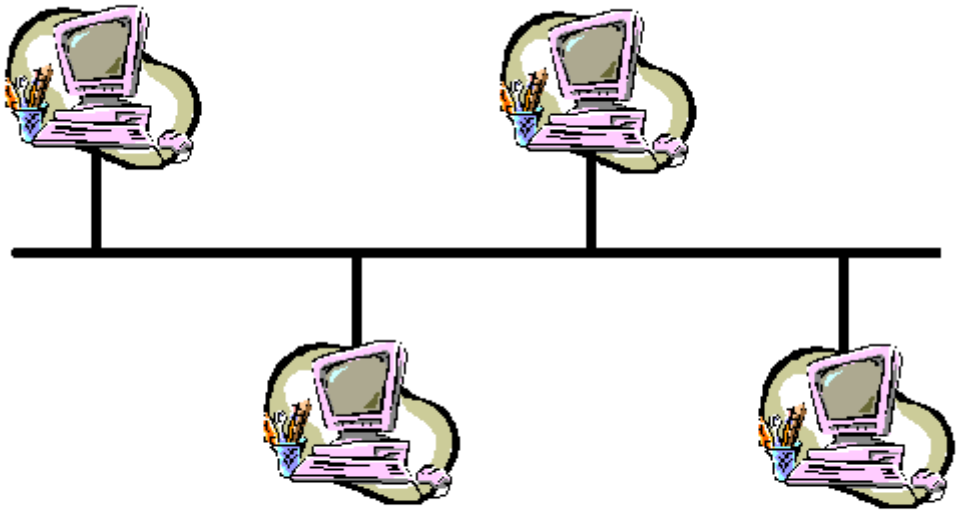
SERVIZIO DISTRIBUITO DI REGISTRAZIONE



Zambellini Stefano
e-mail zambe@libero.it
matr. 2146 57292

SOMMARIO.

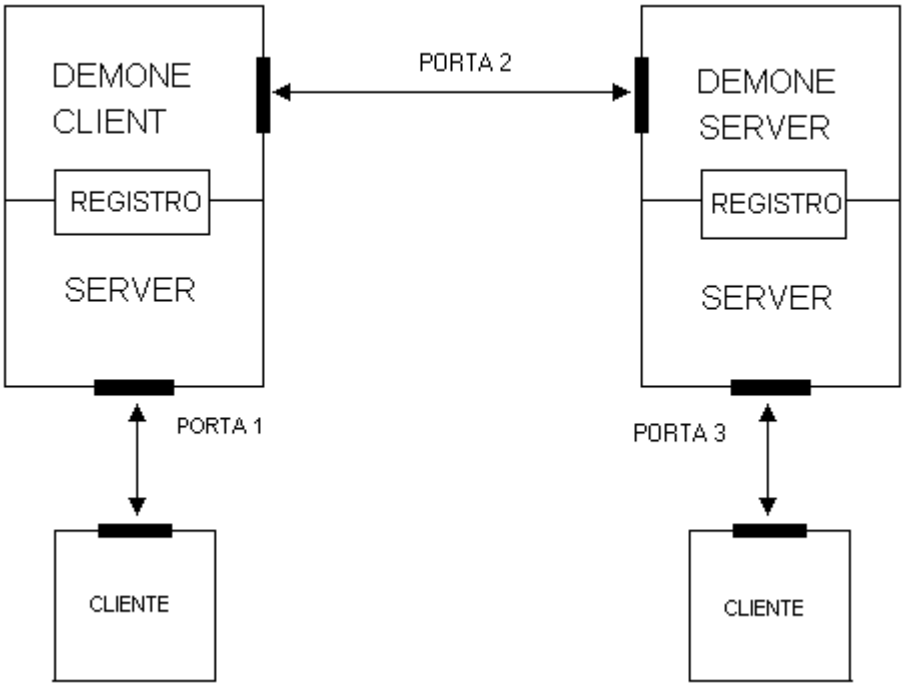
SPECIFICHE.

Realizzazione di un sistema di registrazione distribuito su più macchine al fine di :

1. Ridurre i problemi di congestione.
2. Duplicare il servizio , per dare maggiori garanzie di funzionamento anche in caso di guasto.

Tale sistema prevede un meccanismo di recupero in caso di caduta di un nodo ed inoltre garantisce la consistenza dei dati registrati dai diversi server distribuiti.

SCHEMA E DESCRIZIONE.



Lo scopo è permettere la registrazione di un utente ad un servizio generico.
Il sistema risulta duplicato in due server posti su nodi differenti.
Il cliente può decidere a quale server collegarsi per effettuare la registrazione : i suoi dati verranno memorizzati in un file di registro per essere trattati a posteriori.
Ovviamente esiste un file di registro su ogni nodo, però il sistema garantisce l'uguaglianza di questi, possiamo infatti notare che su ogni server esiste un demone incaricato di dialogare col demone dell'altro server , al fine di garantire la consistenza dei registri.

Ciascun server realizza il suo compito sfruttando il meccanismo dei THREAD , infatti troviamo i seguenti componenti :

- Un THREAD server che accoglie i clienti generando un ulteriore thread che si occupa di ciascuno di essi, si utilizzano quindi server paralleli.
- Un DEMONE che periodicamente si mette in contatto col demone dell'altro server per permettere il passaggio dei dati e garantire la consistenza dei registri(sarà gestito col metodo CLIENT – SERVER sequenziale). Oltre a questo il demone evita la registrazione di nomi doppi.

IPOTESI DI GUASTO.

Durante il funzionamento possono accadere diversi guasti che verranno trattati nel seguente modo:

1. Caduta di un nodo server:
    - Il cliente ad esso collegato andrà fuori servizio : sarà necessario ricollegarsi all'altro server od allo stesso dopo un eventuale recupero.
    - Attuazione di una strategia di recupero che permetta al server di tornare in servizio senza perdere i dati già registrati.
2. Caduta di un cliente :
    - In questo caso non sarà prevista alcuna funzione "ARE YOU ALIVE ? " per poter liberare le risorse dedicate dal server a quel cliente (si presuppone , allora , che questo non avvenga).

CLIENTE.

Il cliente sarà munito di una semplice interfaccia grafica con un campo per inserire il testo da registrare e quindi da inviare al server , ed uno per i messaggi inviati dal server (normalmente sarà l'eco del nostro dato).

Importazione dei package necessari .

```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.GridLayout;
-import java.awt.event.*;
import javax.swing.*;
```

Da questo punto verrà creato un pannello in cui verranno inseriti diversi campi : i due già descritti piu' eventuali intestazioni ed un pulsante di conferma.

```
public class Cliente extends JFrame implements ActionListener {

        private JLabel Tit1 = new JLabel ("Server",SwingConstants.CENTER);
        private JLabel Tit2 = new JLabel ("Cliente",SwingConstants.CENTER);
        private JTextField MsgSvr = new JTextField("Attesa di connessione",20);
        private JTextField MsgClient = new JTextField("",20);
        private JButton pulsante = new JButton("Invia");
        private JLabel Vuoto = new JLabel();
        private static BufferedReader in;
        private static PrintWriter out;
        private static Socket socket;

        public Cliente(String indirizzo ,String porta) {

                super ("Cliente1");

                MsgSvr.setEditable(false);


                JPanel pannello = new JPanel();


                pannello.setLayout(new GridLayout (3,2,5,8));
                pannello.add(Tit1);
                pannello.add(MsgSvr);
                pannello.add(Tit2);
                pannello.add(MsgClient);
                pannello.add(Vuoto);
                pannello.add(pulsante);
```

```
            setContentPane(pannello);
```

Verrà ora creata la socket di comunicazione con l’indirizzo specificato dagli argomenti di ingresso ed i flussi di input ed output per la socket stessa.

```
            try {

                    InetAddress addr =
                    InetAddress.getByName(indirizzo);
                    socket =
                    new Socket(addr, Integer.parseInt(porta));
                    try {
                            in =
                            new BufferedReader(new InputStreamReader(
                            socket.getInputStream()));
                            out =
                            new PrintWriter(
                            new BufferedWriter(new OutputStreamWriter(
                            socket.getOutputStream())),true);
                            MsgSvr.setText("Connessione avvenuta");
                            pulsante.addActionListener(this);
                            repaint();
                            } catch(IOException e) {
                                    MsgSvr.setText("closing...");
                                    socket.close();}
                    } catch(IOException e) {}
    }
```

Main del programma in cui si verificano gli argomenti di ingresso ed in cui si apre l’interfaccia grafica per l’utente.

```
     public static void main(String[] args) {
            if (args.length != 2) {
                    System.out.println("Uso: Cliente indirizzo numeroporta");
                    System.exit(1);
            }
            JFrame frame = new Cliente(args[0],args[1]);
```

In questo punto si gestisce la chiusura della comunicazione che potrà avvenire chiudendo l’interfaccia grafica(piu’ avanti si potrà vedere che lo stesso effetto si ottiene inviando “END”)

```
            WindowListener l = new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {
```

La chiusura si ottiene comunque inviando “END”.

```
                        try{

                            out.println("END");
                            out.close();
                            socket.close();
                            System.exit(0);}  catch(IOException er) {}
                    }
            };

            frame.addWindowListener(l);

            frame.pack();
            frame.setVisible(true);


    }
```

Questa routine gestisce l’evento pressione del pulsante INVIO. Si legge quanto scritto nel campo di testo da inviare e lo si spedisce , attendendo l’eco dal Server stesso.

```
        public void actionPerformed(ActionEvent evt) {

                String messaggio;

                Object source = evt.getSource();

                if (source == pulsante ) {
                   try {messaggio = MsgClient.getText();
                        out.println(messaggio);
                        String str = in.readLine();
                        MsgSvr.setText(""+str);
                        MsgClient.setText("");} catch(IOException e) {};


                }

                repaint();
        }

}
```

SERVER1.

E’ un server con due stati.
Alla partenza genera un thread “Multiserver” il quale accetta i clienti creando per ciascuno di essi un ulteriore thread dedicato (thread “UnServer”).
“UnServer” riceve i dati dal cliente attraverso la PORTA1 (8080) e li salva in un file temporaneo (“temp1.tmp” , se siamo nello stato 0 o “temp2.tmp” , se siamo nello stato1. Di default si parte dallo 0).
Viene creato anche un demone che ha il compito di impostare lo stato di partenza , in caso di recupero da una situazione di quasto.
Questo demone diventa il client del demone generato sul nodo del server2 (Si usa la PORTA2 ,cioè 8081).
Periodicamente si fa spedire il numero di voci registrate nel file temporaneo del server2 a cui somma il numero delle voci del suo .
Se il totale supera un certo valore (“VOCI_TOTALI”) inizierà una fase di trasferimento di dati (aggiornamento del registro).
Il demone si fa trasferire il file temporaneo del server2 e lo memorizza nel file registro (“reg1.dat”), dopo di che memorizza anche il suo file temporaneo spedendolo contemporaneamente all’altro server.
Si chiuderà così la connessione col demone del server2.
Ad ogni aggiornamento del registro si cambia stato cioè si cambia il file temporaneo su cui si scrivono i dati provenienti dal cliente, al fine di evitare di scrivere su un file che si stà copiando in un altro.
Durante l’aggiornamento si verificano le voci già registrate per evitare duplicati.

Si importano i package necessari.

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

Si prepara l’ambiente comune condiviso da tutti i thread :

1. Un file di registro , reg1.
2. Due porte di comunicazione .
   - PORTA1 che serve per collegarsi con il cliente.
   - PORTA2 utilizzata per connettersi col server2.
3. Due file temporanei utilizzati a seconda dello stato.

4. Un contatore (Cont) usato per contare le voci registrate nel file temporaneo corrente.
5. La variabile di stato (stato) di tipo booleano:
   - stato = false , siamo nello stato zero
   - stato = true, siamo nello stato uno
6. Una variabile VOCI_TOTALI dove si imposta il limite di voci dopo il quale occorre fare l'aggiornamento.Infatti, come si è gia detto, periodicamente il Demone del server1 effettua la somma del numero di voci registrate nei file temporanei dei due server e se il totale supera questo limite avviene l'aggiornamento.
7. Una variabile INTERVALLO che specifica ogni quanto tempo (in microsecondi) si verifica se fare l'aggiornamento o meno.

```
public class Server1 extends Thread {
       static PrintWriter reg1;
       static final int PORTA1 = 8080;
       static final int PORTA2 = 8081;
       static final int VOCI_TOTALI = 5;
       static final int INTERVALLO = 5000;
        static PrintWriter temp1;
       static PrintWriter temp2;
       static int Cont=0;
       static String indirizzo_server_2;
       static boolean stato = false;
```

Nel main si verifica l'unico argomento , l'indirizzo del server2; si preparano i flussi di output per i due file temporanei e per il file di registro ; si instanziano gli oggetti "Multiserver" e "Demone";

```
       public static void main (String[] args){

            if (args.length != 1 ) {
                 System.out.println("Uso: Server indirizzo_secondo_Server");
                 System.exit(1);}

                 else indirizzo_server_2 = args[0];


              try {

                       reg1 = new PrintWriter(new BufferedWriter(
                       new FileWriter("reg1.dat",true)));

                       temp1 =
                       new PrintWriter(
                       new BufferedWriter(
                       new FileWriter("Temp1.tmp",true)));

                       temp2 =
                       new PrintWriter(
                       new BufferedWriter(
                       new FileWriter("Temp2.tmp",true)));



                       MultiServer server1 = new MultiServer();
                       server1.start();
                       Demone demone = new Demone();
                       demone.start();
```

Si attende la pressione del testo 'q' per terminare tutto.

```
                        while (System.in.read() != 'q'){}
                        temp1.close();
                        temp2.close();
                        reg1.close();
                        System.exit(0);

                }catch(IOException e){};
        }

        Server1 (String arg) {
                super(arg);
        }

}
```

Si crea la variabile s di tipo Serversocket sulla quale si resta in attesa del collegamento di un cliente.In questo caso ,per ognuno, si instanzia un thread “UnServer” che fornisce il servizio.

```
class MultiServer extends Server1 {



        MultiServer(){
                super("MultiServer");

        }

        public void run() {

        try{

                ServerSocket s = new ServerSocket(PORTA1);
                System.out.println("Server Started");
                try {
                        while(true) {

                                Socket socket = s.accept();
                                try {
                                        new UnServer(socket);
                                } catch(IOException e) {
                                        socket.close();
                                }
                        }
                } finally {
                        s.close();
                }
        }catch(IOException er) {}
        }
}
```

Si creano i flussi di input ed output per la variabile socket passata dal thread “Multiserver” , quindi si entra in un ciclo infinito dove si leggono i dati provenienti dal cliente e li si memorizzano nel file temporaneo corrente (a seconda dello stato). Contemporaneamente si fornisce un servizio di eco.
Il thread termina all’uscita dal ciclo ,cioè quando il cliente spedisce la parola chiave “END”.

```
class UnServer extends MultiServer {
        private Socket socket;
        private BufferedReader in;
        private PrintWriter out;

        UnServer(Socket s)
        throws IOException {
                        super();
                        socket = s;
```

```
                        in =
                        new BufferedReader(
                        new InputStreamReader(
                        socket.getInputStream()));


                        out =
                        new PrintWriter(
                        new BufferedWriter(
                        new OutputStreamWriter(

socket.getOutputStream())), true);
                        start();
        }
        public void run() {
                try {
                        while (true) {
                                String str = in.readLine();
                                if (str.equals("END")) break;
                                System.out.println("Echoing: " + str);
```

Se siamo nello stato 0 il file temporaneo è temp1 (“temp1.tmp), se siamo nello stato 1 il file temporaneo e temp2 (“temp2.tmp”).

```
                                if (!stato){ temp1.println(str);
                                                temp1.flush();}
                                         else { temp2.println(str);
                                                temp2.flush();}
                                Cont++;
                                out.println(str);
                        }
                        System.out.println("closing...");

                } catch (IOException e) {}
                finally {
                        try {
                                socket.close();
                        } catch(IOException er) {}
                        }
        }


}
```

Nel demone come prima cosa si preparano le variabili da utilizzare, quindi nel costruttore si dichiara che il thread è un demone.

```
class Demone extends Server1 {

   private BufferedReader temp1_copy;
   private BufferedReader sockin;
   private PrintWriter sockout;
   private Socket socket ;
   private int Contstato1 = 0 ;
   private int Contstato0 = 0 ;
   private int Cont2;
   private String info;
   private InetAddress addr;

   Demone (){
      super("Demone");
      setDaemon(true);
```

Si leggono i due file temporanei per contare il numero di voci che posseggono.In uno stato consistente un solo file posiede voci, l’altro dovrebbe essere vuoto o al limite entrambi sono vuoti.

```
      try{

                FileReader tempread = new FileReader ("temp1.tmp");
                BufferedReader temp = new BufferedReader (tempread);
```

```
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato0++;
                }
                temp.close();
        } catch (IOException e){}

      try{


                FileReader tempread = new FileReader ("temp2.tmp");
                BufferedReader temp = new BufferedReader (tempread);
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato1++;
                }
                temp.close();
        } catch (IOException e){}
```

Si confrontano i dati ottenuti dalle letture precedenti per impostare lo stato di partenza (e quindi il file temporaneo corrente).Se entrambi i file non sono vuoti si è in uno stato inconsistente cioè errato, quindi si esce(Se tutto funziona correttamente tale stato non si dovrebbe mai raggiungere).

```
if ((Contstato0 == 0) && (Contstato1 == 0)) {
                                        Cont=0;
                                        stato = false;
                                        }

if ((Contstato0 > 0) && (Contstato1 == 0)) {
                                        Cont=Contstato0;
                                        stato = false;
                                        }

if ((Contstato0 == 0) && (Contstato1 > 0)) {
                                        Cont=Contstato1;
                                        stato = true;
                                        }

if ((Contstato0 > 0) && (Contstato1 > 0)) {
                                        System.out.println("Errore: stato inconsistente");
                                        System.exit(1);
                                        }


   }


  public void run(){
```

Come prima cosa ci si pone in un ciclo infinito.

```
    while(true){
```

Una volta al ciclo si attende per un certo numero di microsecondi.

```
      try {
          Thread.sleep(INTERVALLO);
          }catch (InterruptedException e){System.out.println(" Sleep non funzionante");}
```

Si ottiene l’indirizzo del sel sever2 e ci si collega attraverso la PORTA2.

```
               try {
```

```
addr = InetAddress.getByName(indirizzo_server_2);
socket = new Socket(addr, PORTA2);
System.out.println ("\nConnessione col Server2 avvenuta");
```

Si preparano i flussi di input ed output per la socket.

```
try {
        sockin =
        new BufferedReader(new InputStreamReader(
        socket.getInputStream()));
        sockout =
        new PrintWriter(
        new BufferedWriter(new OutputStreamWriter(
        socket.getOutputStream())),true);
```

Si verifica se la somma del contatore locale con quello spedito dal server2 supera un certo valore, incaso affermativo si procede spedendo all'altro server il proprio contatore , altrimenti si spedisce –1 e si ricomincia il ciclo.

```
// Scambio dati Server1 <-> Server2


Cont2 = Integer.parseInt(sockin.readLine());

System.out.println("Contatore Server1 = "+Cont);
System.out.println("Contatore Server2 = "+Cont2);


  if ((Cont+Cont2) >= VOCI_TOTALI) {
```

Si cambia immediatamente stato per poter lavorare su dei file non in uso nel thread "Unserver".

```
      stato = !stato;

      sockout.println(Integer.toString(Cont));
      if (stato) {
```

Si leggono i dati provenienti dal server2, si verificano che non siano già stati memorizzati nel registro, quindi si memorizzano in quest'ultimo.

```
            temp1_copy = new BufferedReader (new FileReader("temp1.tmp"));
          System.out.println("Dati Server2");
          for (; Cont2 > 0 ; Cont2--){
              info = sockin.readLine();
              if (!verifica(info)) {
              reg1.println(info);
              System.out.println(info);
              reg1.flush();
              }}
```

Si leggono i dati dal file temporaneo locale , si spediscono al server2 e si memorizzano nel registro , verificando che non siano già presenti.

```
      System.out.println("Dati Server1");
          for (;Cont > 0 ;Cont--){
                        info = temp1_copy.readLine();
                        sockout.println(info);
                        if (!verifica(info)) {
                        reg1.println(info);
                        System.out.println(info);
                        reg1.flush();}}
```

Si azzera il file temporaneo appena memorizzato.

```
                    temp1.close();
```

```
                    temp1 = new PrintWriter(new BufferedWriter(
                    new FileWriter("Temp1.tmp",false)));
                    temp1_copy.close();
                    }
```

Si rifanno le stesse operazioni appena descritte ma nel caso di stato diverso.

```
       else {
              temp1_copy = new BufferedReader (new FileReader("temp2.tmp"));

          System.out.println("Dati Server2");
          for (; Cont2 > 0 ; Cont2--){
              info = sockin.readLine();
              if (!verifica(info)) {
              reg1.println(info);
              System.out.println(info);
              reg1.flush();
              }}
          System.out.println("Dati Server1");
          for (;Cont > 0 ;Cont--){
                              info = temp1_copy.readLine();
                              sockout.println(info);
                              if (!verifica(info)) {
                              reg1.println(info);
                              System.out.println(info);
                              reg1.flush();}}

                    temp2.close();
                    temp2 = new PrintWriter(new BufferedWriter(
                    new FileWriter("Temp2.tmp",false)));
                    temp1_copy.close();
                    }
  }
  else sockout.println(-1);


                         } finally {

                         System.out.println("Sconnessione dal Server2");
                         socket.close();}

                      } catch (IOException err) {
                                  System.out.println ("Non c'e il Server2");}

                 }

  }
```

Funzione di verifica: il parametro passato viene confrontato con le voci già memorizzate nel registro.

```
  private boolean verifica (String nome){

      try{
                BufferedReader reg = new BufferedReader (
                                        new FileReader("reg1.dat"));

                boolean eof = false;
                while (!eof){
                        String line = reg.readLine();

                        if (line == null)
                                eof = true;
                                else if (line.equals(nome)) {reg.close();
                                                             return (true) ;}

                }
         reg.close();

         } catch (IOException e) {}
     return (false);

  }
```

```
}
```

SERVER2.

Il server2 è molto simile al server1 ad eccezione del demone che fra le altre cose svolge una funzione di server, quindi non occorre alcun argomento d'ingresso.
Ovviamente , come si può capire, tale demone risulta essere il complementare di quello del server1, quindi, alla connessione, spedisce il proprio contatore , riceve quello dell'altro in caso di aggiornamento oppure –1 se si deve ricominciare da capo, memorizza il file temporaneo sul proprio registro e lo spedisce all'altro server, riceve il file temporaneo dell'altro server e lo memorizza.

Nel demone vengono fatte le solite letture e verifiche per decidere lo stato di partenza.

```
class Demone2 extends Server2 {

   private BufferedReader temp ;
   private BufferedReader temp1_copy;
   private BufferedReader sockin;
   private PrintWriter sockout;
   private ServerSocket socket ;
   private Socket sock;
   private int Contstato1 = 0 ;
   private int Contstato0 = 0 ;
   private String info;
   private int Cont2 = 0;

   Demone2 (){
      super("Demone");
      setDaemon(true);

      try{


                temp = new BufferedReader (new FileReader ("temp3.tmp"));
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato0++;
                }
                temp.close();
        } catch (IOException e){}

      try{


                temp = new BufferedReader (new FileReader ("temp4.tmp"));
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato1++;
                }
                temp.close();
        } catch (IOException e){}


if ((Contstato0 == 0) && (Contstato1 == 0)) {
                                        Cont=0;
                                        stato = false;
                                        }

if ((Contstato0 > 0) && (Contstato1 == 0)) {
                                        Cont=Contstato0;
                                        stato = false;
```

```
                                        }

if ((Contstato0 == 0) && (Contstato1 > 0)) {
                                        Cont=Contstato1;
                                        stato = true;
                                        }

if ((Contstato0 > 0) && (Contstato1 > 0)) {
                                        System.out.println("Errore: stato inconsistente");
                                        System.exit(1);
                                        }


   }

  public void run(){
```

Si crea la ServerSocket di accettazione e si entra in un ciclo infinito nel quale si creano i flussi di input ed output per la socket di comunicazione.

```
    try{
          ServerSocket socket = new ServerSocket(PORTA2);


        while (true){
           try {

               Socket sock = socket.accept();
               try {
                  System.out.println("\nConnessione col Server1 avvenuta");


  BufferedReader sockin = new BufferedReader( new InputStreamReader(
                                      sock.getInputStream()));

  PrintWriter sockout = new PrintWriter( new BufferedWriter(
                new OutputStreamWriter(sock.getOutputStream())),true);
```

Si spedisce il contatore locale al server1 e si attende la risposta.Se si ottiene –1 si ricomincia il ciclo altrimenti si prosegue col trasferimento di dati.

```
sockout.println(Integer.toString(Cont));
Cont2 = Integer.parseInt(sockin.readLine());


 if (Cont2 > -1 ) {
    System.out.println("Contatore Server1 = " + Cont2);
    System.out.println("Contatore Server2 = " + Cont);
    stato = !stato;
```

Contemporaneamente si spedisce il proprio file temporaneo al server1 e lo si memorizza nel registro.
Anche in questo caso si fanno le opportune verifiche per evitare di duplicare delle voci nel registro.

```
    if (stato){
        System.out.println("Dati Server2");
        temp1_copy = new BufferedReader (new FileReader ("temp3.tmp"));
        for(; Cont > 0 ;Cont--){
                info=temp1_copy.readLine();
                sockout.println(info);
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}
```

Si riceve il file temporaneo dal server1 e lo si memorizza nel registro effettuando le dovute verifiche.

```
        System.out.println("Dati Server1");
        for(; Cont2 > 0 ;Cont2--){
                info=sockin.readLine();
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}
```

Si azzera il file temporaneo appena usato.

```
                        temp1.close();
                        temp1 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp3.tmp",false)));
                temp1_copy.close();
    }
```

Si rifanno le stesse operazioni di prima ma nel caso in cui lo stato sia diverso.

```
    else {
        temp1_copy = new BufferedReader (new FileReader ("temp4.tmp"));
        System.out.println("Dati Server2");
        for(; Cont > 0 ;Cont--){
                info=temp1_copy.readLine();
                sockout.println(info);
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}
        System.out.println("Dati Server1");
        for(; Cont2 > 0 ;Cont2--){
                info=sockin.readLine();
                if (!verifica(info)){
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}

                        temp2.close();
                        temp2 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp4.tmp",false)));
                temp1_copy.close();

     }

} else
        System.out.println("Tarsferimento non avvenuto : sarebbe inefficente");



               } finally {
               System.out.println("Sconnessione dal Server1");
               sock.close();}

           } catch (IOException er) {System.out.println ("Socket non accettata");}

       }

     } catch (IOException err) { System.out.println("Non Started");}

}
```

NOTA INTEGRATIVA.

Dai listati si capisce che ciascun server , prima di aggiornare il registro, effettua una verifica su tutti i nomi già memorizzati per evitare di inserire doppioni.
Per esempio se si fanno le seguenti ipotesi

1. Il registro ha già N voci registrate
2. Ad ogni aggiornamento del registro si hanno in media K nomi nel file temporaneo

allora all R-esimo aggiornamento si effettueranno P verifiche.

$$P = KN + K(N + K) + K(N + 2K) + ... + K(N + (R-1)K)$$

$$P = RKN + \sum_{i=0}^{R-1} iK^2 = RKN + R\frac{R-1}{2}K^2$$

PARTICOLARI ACCORGIMENTI.

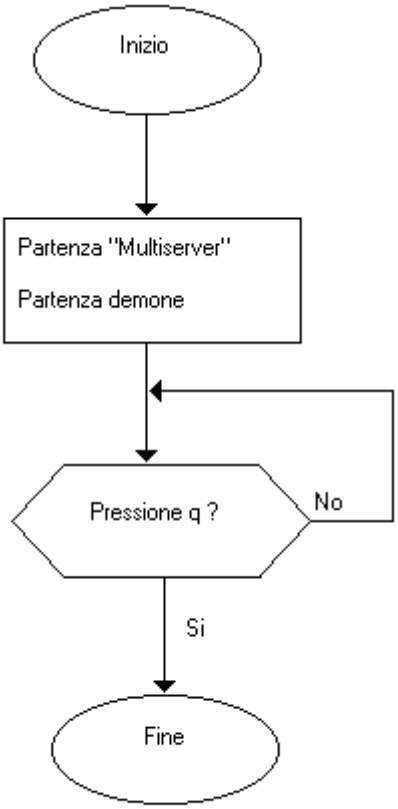
Per evitare di impegnare troppo la risorsa “CANALE” si effettua una connessione diversa ad ogni aggiornamento sconnettendosi alla fine.
In questo modo risulta anche più facile il recupero in caso di guasto di un server.

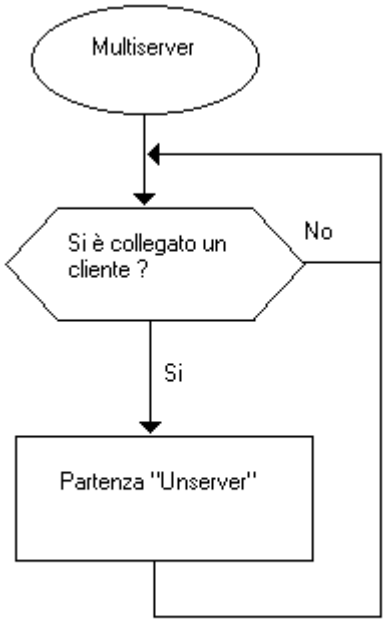
STRATEGIA DI RECUPERO.

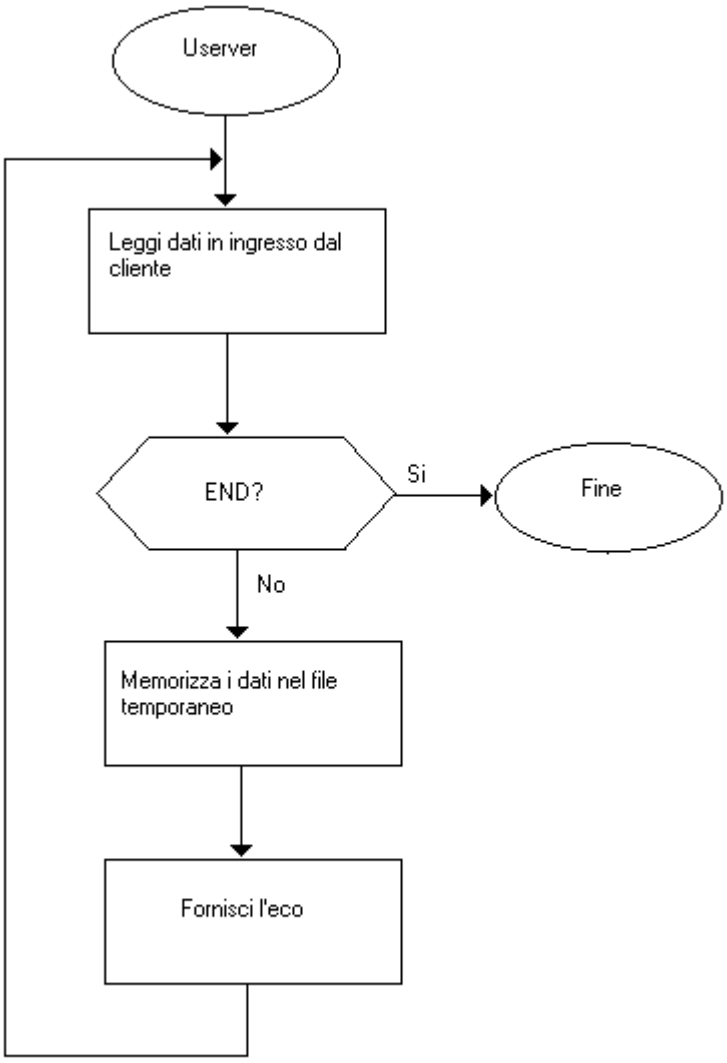
Come già spiegato, i demoni dei due server aprono una connessione nuova ogni “INTERVALLO” microsecondi , chiudendola alla fine della verifica o dell’aggiornamento.
In questo modo , fra una connessone e la successiva, se uno dei due server cade l’altro continua a fornire il servizio lavorando solo sul proprio file temporaneo.
Una volta riattivato il server caduto la connessione potrà essere effettuata e tutto tornerà come prima con eventuali aggiornamenti.
Anche durante la connessione un server può cadere a patto che l’altro non sia in fase di lettura altrimeni quest’ultimo rimarrebbe bloccato.
Il server però si troverà in questa fase per una frazione molto piccola del tempo totale fra una connessione e la successiva , quindi la probbabilità di tale evento risulta piuttosto bassa e accettabile.

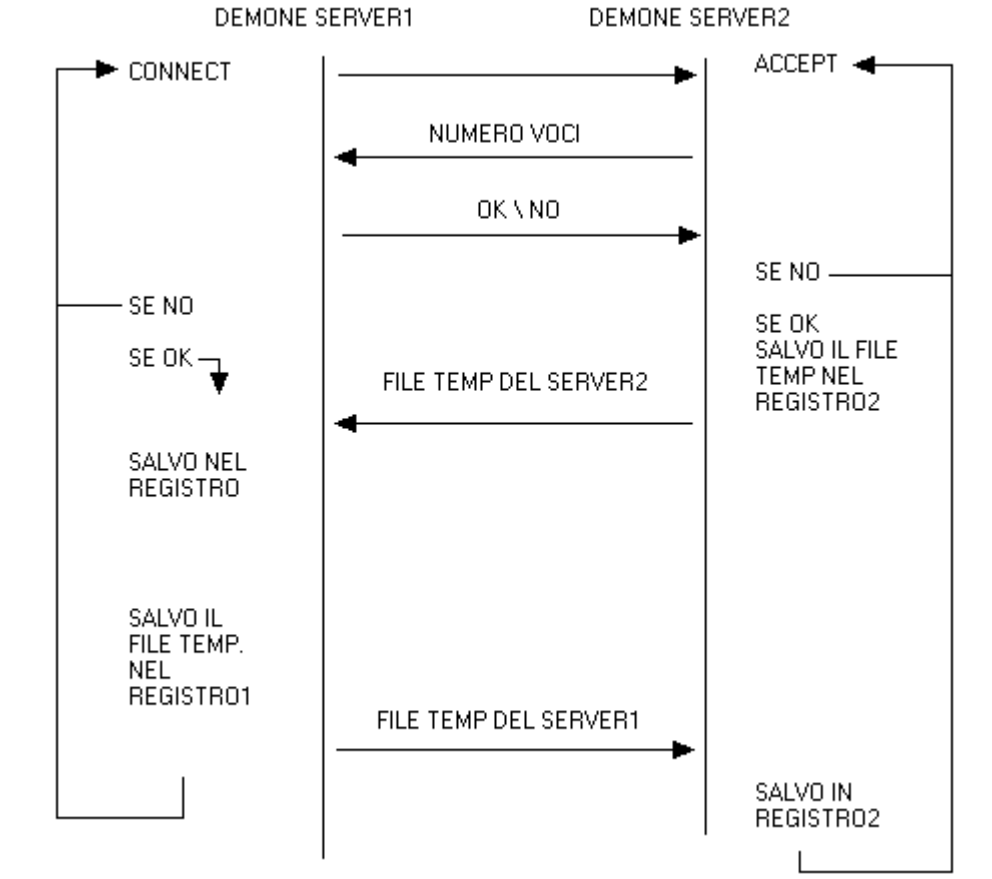
DIAGRAMMI DI FLUSSO DEI COMPONENTI.

Come già spiegato la parte di server che interagisce col cliente è praticamente uguale sia in server1 che in server2.
Alla fine si può vedere lo schema di comunicazione fra i demoni dei due server.


Inizio
Partenza "Multiserver"
Partenza demone
Pressione q ?
No
Si
Fine


Multiserver
Si è collegato un cliente ?
No
Si
Partenza "Unserver"


Userver
Leggi dati in ingresso dal cliente
END?
Si
Fine
No
Memorizza i dati nel file temporaneo
Fornisci l'eco



MANUALE D'USO.

1. Lanciare il Server1 : "java Server1 indirizzo_Server2"
2. Lanciare il Server2 : "java Server2"
3. Lanciare il Cliente : " java Cliente indirizzo_server_desiderato porta_server_desiderato

Se i server fossero su macchine diverse si potrebbe usare la stessa porta poiché , essendo diversi gli indirizzi, sarebbero diverse le connessioni.
Però, per ragioni di collaudo (su un'unica macchina), l'indirizzo dei server è il medesimo, quindi per realizzare connessioni diverse servono porte diverse.
Inoltre, grazie alle strategie di recovery implementate, è indifferente eseguire prima il passo 1 od il passo 2.

ESEMPIO.

Come provare il sistema su un' unica macchina.

1. java Server2
2. java Server1 localhost
3. Per collegarsi al Server1 : java Cliente localhost 8080
4. Per collegarsi al Server2 : java Cliente localhost 8082

LISTATO CLIENTE.

```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

public class Cliente extends JFrame implements ActionListener {

        private JLabel Tit1 = new JLabel ("Server",SwingConstants.CENTER);
        private JLabel Tit2 = new JLabel ("Cliente",SwingConstants.CENTER);
        private JTextField MsgSvr = new JTextField("Attesa di connessione",20);
        private JTextField MsgClient = new JTextField("",20);
        private JButton pulsante = new JButton("Invia");
        private JLabel Vuoto = new JLabel();
        private static BufferedReader in;
        private static PrintWriter out;
        private static Socket socket;

        public Cliente(String indirizzo ,String porta) {

                super ("Cliente1");

                MsgSvr.setEditable(false);


                JPanel pannello = new JPanel();


                pannello.setLayout(new GridLayout (3,2,5,8));
                pannello.add(Tit1);
                pannello.add(MsgSvr);
                pannello.add(Tit2);
                pannello.add(MsgClient);
                pannello.add(Vuoto);
                pannello.add(pulsante);
                setContentPane(pannello);

                try {

                        InetAddress addr =
                        InetAddress.getByName(indirizzo);
                        socket =
                        new Socket(addr, Integer.parseInt(porta));
                        try {
                                in =
                                new BufferedReader(new InputStreamReader(
                                socket.getInputStream()));
                                out =
                                new PrintWriter(
                                new BufferedWriter(new OutputStreamWriter(
                                socket.getOutputStream())),true);
                                MsgSvr.setText("Connessione avvenuta");
                                pulsante.addActionListener(this);
                                repaint();
                                } catch(IOException e) {
                                        MsgSvr.setText("closing...");
                                        socket.close();}
                        } catch(IOException e) {}
        }

        public static void main(String[] args) {
                if (args.length != 2) {
                        System.out.println("Uso: Cliente indirizzo numeroporta");
                        System.exit(1);
                }
                JFrame frame = new Cliente(args[0],args[1]);
                WindowListener l = new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                            try{
```

```
                                out.println("END");
                                out.close();
                                socket.close();
                                System.exit(0);}  catch(IOException er) {}
                        }
                };

                frame.addWindowListener(l);

                frame.pack();
                frame.setVisible(true);


        }

        public void actionPerformed(ActionEvent evt) {

                String messaggio;

                Object source = evt.getSource();

                if (source == pulsante ) {
                   try {messaggio = MsgClient.getText();
                        out.println(messaggio);
                        String str = in.readLine();
                        MsgSvr.setText(""+str);
                        MsgClient.setText("");} catch(IOException e) {};


                }

                repaint();
        }

}
```

LISTATO SERVER1.

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



public class Server1 extends Thread {
        static PrintWriter reg1;
        static final int PORTA1 = 8080;
        static final int PORTA2 = 8081;
        static final int VOCI_TOTALI = 5;
        static final int INTERVALLO = 5000;
         static PrintWriter temp1;
        static PrintWriter temp2;
        static int Cont=0;
        static String indirizzo_server_2;
        static boolean stato = false;

        public static void main (String[] args){

             if (args.length != 1 ) {
                  System.out.println("Uso: Server indirizzo_secondo_Server");
                  System.exit(1);}

                  else indirizzo_server_2 = args[0];


                try {

                        reg1 = new PrintWriter(new BufferedWriter(
                        new FileWriter("reg1.dat",true)));

                        temp1 =
                        new PrintWriter(
```

```
                        new BufferedWriter(
                        new FileWriter("Temp1.tmp",true)));

                        temp2 =
                        new PrintWriter(
                        new BufferedWriter(
                        new FileWriter("Temp2.tmp",true)));



                        MultiServer server1 = new MultiServer();
                        server1.start();
                        Demone demone = new Demone();
                        demone.start();

                        while (System.in.read() != 'q'){}
                        temp1.close();
                        temp2.close();
                        reg1.close();
                        System.exit(0);

                }catch(IOException e){};
        }

        Server1 (String arg) {
                super(arg);
        }

}


class MultiServer extends Server1 {



        MultiServer(){
                super("MultiServer");

        }

        public void run() {

        try{

                ServerSocket s = new ServerSocket(PORTA1);
                System.out.println("Server Started");
                try {
                        while(true) {

                                Socket socket = s.accept();
                                try {
                                        new UnServer(socket);
                                } catch(IOException e) {
                                        socket.close();
                                }
                        }
                } finally {
                        s.close();
                }
        }catch(IOException er) {}
        }
}


class UnServer extends MultiServer {
        private Socket socket;
        private BufferedReader in;
        private PrintWriter out;

        UnServer(Socket s)
        throws IOException {
                        super();
                        socket = s;
                        in =
                        new BufferedReader(
                        new InputStreamReader(
```

```
                        socket.getInputStream()));


                        out =
                        new PrintWriter(
                        new BufferedWriter(
                        new OutputStreamWriter(

                        socket.getOutputStream())), true);
                        start();
        }
        public void run() {
                try {
                        while (true) {
                                String str = in.readLine();
                                if (str.equals("END")) break;
                                System.out.println("Echoing: " + str);

                                if (!stato){ temp1.println(str);
                                                temp1.flush();}
                                         else { temp2.println(str);
                                                temp2.flush();}
                                Cont++;
                                out.println(str);
                        }
                        System.out.println("closing...");

                } catch (IOException e) {}
                finally {
                        try {
                                socket.close();
                        } catch(IOException er) {}
                        }
        }



}


class Demone extends Server1 {

   private BufferedReader temp1_copy;
   private BufferedReader sockin;
   private PrintWriter sockout;
   private Socket socket ;
   private int Contstato1 = 0 ;
   private int Contstato0 = 0 ;
   private int Cont2;
   private String info;
   private InetAddress addr;

   Demone (){
      super("Demone");
      setDaemon(true);

      try{

                FileReader tempread = new FileReader ("temp1.tmp");
                BufferedReader temp = new BufferedReader (tempread);
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato0++;
                }
                temp.close();
        } catch (IOException e){}

      try{


                FileReader tempread = new FileReader ("temp2.tmp");
                BufferedReader temp = new BufferedReader (tempread);
                boolean eof = false;
                while (!eof){
```

```
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato1++;
                }
                temp.close();
        } catch (IOException e){}


if ((Contstato0 == 0) && (Contstato1 == 0)) {
                                        Cont=0;
                                        stato = false;
                                        }

if ((Contstato0 > 0) && (Contstato1 == 0)) {
                                        Cont=Contstato0;
                                        stato = false;
                                        }

if ((Contstato0 == 0) && (Contstato1 > 0)) {
                                        Cont=Contstato1;
                                        stato = true;
                                        }

if ((Contstato0 > 0) && (Contstato1 > 0)) {
                                        System.out.println("Errore: stato inconsistente");
                                        System.exit(1);
                                        }


   }

  public void run(){

     while(true){

       try {
           Thread.sleep(INTERVALLO);
           }catch (InterruptedException e){System.out.println(" Sleep non funzionante");}
                   try {

                        addr = InetAddress.getByName(indirizzo_server_2);
                        socket = new Socket(addr, PORTA2);
                        System.out.println ("\nConnessione col Server2 avvenuta");
                        try {
                                sockin =
                                new BufferedReader(new InputStreamReader(
                                socket.getInputStream()));
                                sockout =
                                new PrintWriter(
                                new BufferedWriter(new OutputStreamWriter(
                                socket.getOutputStream())),true);


// Scambio dati Server1 <-> Server2


Cont2 = Integer.parseInt(sockin.readLine());

System.out.println("Contatore Server1 = "+Cont);
System.out.println("Contatore Server2 = "+Cont2);


  if ((Cont+Cont2) >= VOCI_TOTALI) {
       stato = !stato;

       sockout.println(Integer.toString(Cont));
       if (stato) {


               temp1_copy = new BufferedReader (new FileReader("temp1.tmp"));
           System.out.println("Dati Server2");
           for (; Cont2 > 0 ; Cont2--){
               info = sockin.readLine();
               if (!verifica(info)) {
               reg1.println(info);
               System.out.println(info);
```

```
            reg1.flush();
            }}
         System.out.println("Dati Server1");
         for (;Cont > 0 ;Cont--){
                              info = temp1_copy.readLine();
                              sockout.println(info);
                              if (!verifica(info)) {
                              reg1.println(info);
                              System.out.println(info);
                              reg1.flush();}}


                        temp1.close();
                        temp1 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp1.tmp",false)));
                        temp1_copy.close();
                        }
      else {
             temp1_copy = new BufferedReader (new FileReader("temp2.tmp"));

         System.out.println("Dati Server2");
         for (; Cont2 > 0 ; Cont2--){
             info = sockin.readLine();
             if (!verifica(info)) {
             reg1.println(info);
             System.out.println(info);
             reg1.flush();
             }}
         System.out.println("Dati Server1");
         for (;Cont > 0 ;Cont--){
                              info = temp1_copy.readLine();
                              sockout.println(info);
                              if (!verifica(info)) {
                              reg1.println(info);
                              System.out.println(info);
                              reg1.flush();}}

                        temp2.close();
                        temp2 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp2.tmp",false)));
                        temp1_copy.close();
                        }
}
else sockout.println(-1);


                            } finally {

                            System.out.println("Sconnessione dal Server2");
                            socket.close();}

                       } catch (IOException err) {
                                   System.out.println ("Non c'e il Server2");}

                  }

}
private boolean verifica (String nome){

    try{
              BufferedReader reg = new BufferedReader (
                                   new FileReader("reg1.dat"));

              boolean eof = false;
              while (!eof){
                      String line = reg.readLine();

                      if (line == null)
                              eof = true;
                              else if (line.equals(nome)) {reg.close();
                                                           return (true) ;}

              }
       reg.close();

       } catch (IOException e) {}
   return (false);
```

```
  }

}
```

LISTATO SERVER2.

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



public class Server2 extends Thread {
        static PrintWriter reg2;
        static final int PORTA3 = 8082;
        static final int PORTA2 = 8081;
        static PrintWriter temp1;
        static PrintWriter temp2;
        static int Cont=0;

        static boolean stato = false;

        public static void main (String[] args){

             if (args.length != 0 ) {
                System.out.println("Uso: Server ");
                System.exit(1);}



               try {

                        reg2 = new PrintWriter(new BufferedWriter(
                        new FileWriter("reg2.dat",true)));

                        temp1 =
                        new PrintWriter(
                        new BufferedWriter(
                        new FileWriter("Temp3.tmp",true)));

                        temp2 =
                        new PrintWriter(
                        new BufferedWriter(
                        new FileWriter("Temp4.tmp",true)));



                        MultiServer2 server2 = new MultiServer2();
                        server2.start();
                        Demone2 demone = new Demone2();
                        demone.start();

                        while (System.in.read() != 'q'){}
                        temp1.close();
                        temp2.close();
                        reg2.close();
                        System.exit(0);

                }catch(IOException e){};
        }

        Server2 (String arg) {
                super(arg);
        }

}


class MultiServer2 extends Server2 {
```

```
        MultiServer2(){
                super("MultiServer");

        }

        public void run() {

        try{

                ServerSocket s = new ServerSocket(PORTA3);
                System.out.println("Server Started");
                try {
                        while(true) {

                                Socket socket = s.accept();
                                try {
                                        new UnServer2(socket);
                                } catch(IOException e) {
                                        socket.close();
                                }
                        }
                } finally {
                        s.close();
                }
        }catch(IOException er) {}
        }
}


class UnServer2 extends MultiServer2 {
        private Socket socket;
        private BufferedReader in;
        private PrintWriter out;

        UnServer2(Socket s)
        throws IOException {
                        super();
                        socket = s;
                        in =
                        new BufferedReader(
                        new InputStreamReader(
                        socket.getInputStream()));


                        out =
                        new PrintWriter(
                        new BufferedWriter(
                        new OutputStreamWriter(
                        socket.getOutputStream())), true);
                        start();
        }
        public void run() {
                try {
                        while (true) {
                                String str = in.readLine();
                                if (str.equals("END")) break;
                                System.out.println("Echoing: " + str);

                                if (!stato){ temp1.println(str);
                                                temp1.flush();}
                                         else { temp2.println(str);
                                                temp2.flush();}
                                Cont++;
                                out.println(str);
                        }
                        System.out.println("closing...");

                } catch (IOException e) {}
                finally {
                        try {
                                socket.close();
                        } catch(IOException er) {}
                        }
        }
```

```
}


class Demone2 extends Server2 {

   private BufferedReader temp ;
   private BufferedReader temp1_copy;
   private BufferedReader sockin;
   private PrintWriter sockout;
   private ServerSocket socket ;
   private Socket sock;
   private int Contstato1 = 0 ;
   private int Contstato0 = 0 ;
   private String info;
   private int Cont2 = 0;

   Demone2 (){
      super("Demone");
      setDaemon(true);

      try{


                temp = new BufferedReader (new FileReader ("temp3.tmp"));
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato0++;
                }
                temp.close();
        } catch (IOException e){}

      try{


                temp = new BufferedReader (new FileReader ("temp4.tmp"));
                boolean eof = false;
                while (!eof){
                        String line = temp.readLine();

                        if (line == null)
                                eof = true;
                                else Contstato1++;
                }
                temp.close();
        } catch (IOException e){}


if ((Contstato0 == 0) && (Contstato1 == 0)) {
                                        Cont=0;
                                        stato = false;
                                        }

if ((Contstato0 > 0) && (Contstato1 == 0)) {
                                        Cont=Contstato0;
                                        stato = false;
                                        }

if ((Contstato0 == 0) && (Contstato1 > 0)) {
                                        Cont=Contstato1;
                                        stato = true;
                                        }

if ((Contstato0 > 0) && (Contstato1 > 0)) {
                                        System.out.println("Errore: stato inconsistente");
                                        System.exit(1);
                                        }


   }

  public void run(){
```

```
      try{
          ServerSocket socket = new ServerSocket(PORTA2);


        while (true){
           try {

                Socket sock = socket.accept();
                try {
                   System.out.println("\nConnessione col Server1 avvenuta");


  BufferedReader sockin = new BufferedReader( new InputStreamReader(
                                        sock.getInputStream()));

  PrintWriter sockout = new PrintWriter( new BufferedWriter(
                new OutputStreamWriter(sock.getOutputStream())),true);


sockout.println(Integer.toString(Cont));
Cont2 = Integer.parseInt(sockin.readLine());


 if (Cont2 > -1 ) {
    System.out.println("Contatore Server1 = " + Cont2);
    System.out.println("Contatore Server2 = " + Cont);
    stato = !stato;


    if (stato){
        System.out.println("Dati Server2");
        temp1_copy = new BufferedReader (new FileReader ("temp3.tmp"));
        for(; Cont > 0 ;Cont--){
                info=temp1_copy.readLine();
                sockout.println(info);
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}

        System.out.println("Dati Server1");
        for(; Cont2 > 0 ;Cont2--){
                info=sockin.readLine();
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}

                        temp1.close();
                        temp1 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp3.tmp",false)));
                temp1_copy.close();




    }
    else {
        temp1_copy = new BufferedReader (new FileReader ("temp4.tmp"));
        System.out.println("Dati Server2");
        for(; Cont > 0 ;Cont--){
                info=temp1_copy.readLine();
                sockout.println(info);
                if (!verifica(info)) {
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
                }}
        System.out.println("Dati Server1");
        for(; Cont2 > 0 ;Cont2--){
                info=sockin.readLine();
                if (!verifica(info)){
                reg2.println(info);
                reg2.flush();
                System.out.println(info);
```

```
                }}

                        temp2.close();
                        temp2 = new PrintWriter(new BufferedWriter(
                        new FileWriter("Temp4.tmp",false)));
                temp1_copy.close();

     }

} else
        System.out.println("Tarsferimento non avvenuto : sarebbe inefficente");



               } finally {
               System.out.println("Sconnessione dal Server1");
               sock.close();}

           } catch (IOException er) {System.out.println ("Socket non accettata");}

       }

     } catch (IOException err) { System.out.println("Non Started");}

  }
  private boolean verifica (String nome){

      try{
                BufferedReader reg = new BufferedReader (
                                        new FileReader("reg2.dat"));

                boolean eof = false;
                while (!eof){
                        String line = reg.readLine();

                        if (line == null)
                                eof = true;
                                else if (line.equals(nome)) {reg.close();
                                                             return (true) ;}

                }
         reg.close();
         } catch (IOException e) {}
     return (false);

  }



}
```