

Reti di Calcolatori T

Esercitazione 2

GRUPPO 5:

Stefano Lentini
Davide Perinti
Leone Ruggiero
Marco Turzi
Andrea Zanni

- La soluzione adottata utilizza le classi `java.net.Socket` e `java.net.ServerSocket` sfruttano il protocollo TCP per rendere la trasmissione dei files reliable e ordinato.
- Il Server agisce in parallelo creando un Thread per ogni connessione Cliente/Servitore, che gli permette così di rimettersi subito in attesa di una nuova richiesta.

Client(1): Check argomenti

```
if(args.length!=3) {  
    System.out.println("usage: Client min_size(intero) serverAddr serverPort(intero)");  
    System.exit(1);  
}  
try {  
    min_size=Integer.parseInt(args[0]);  
  
    if(min_size<=0) {  
        System.err.println("min_size deve essere >0");  
        System.exit(1);  
    }  
  
    serverPort=Integer.parseInt(args[2]);  
    addr=InetAddress.getByName(args[1]);  
}  
catch(UnknownHostException e) {  
    e.printStackTrace();  
    System.out.println("Remote-host sconosciuto");  
    System.exit(2);  
}  
catch(NumberFormatException e) {  
    e.printStackTrace();  
    System.out.println("usage: Client min_size(intero) serverAddr serverPort(intero)");  
    System.exit(3);  
}
```

Ci aspettiamo che la dimensione minima richiesta sia un valore positivo.

Nel caso il parametro serverAddr non venga riconosciuto come InetAddress.

Errore formato dei valori della porta e della min_size.

Client(2): Ciclo invio Directorys

```
do{
```

```
//richiesta dir e invio contenuto dir  
System.out.println("Inserire il percorso assoluto o relativo della directory di cui si vuole inviare il contenuto (EOF per terminare)");
```

```
try {
```

```
    read_dir=in.readLine();
```

```
    if(read_dir!=null) {
```

```
        File dir=new File(read_dir);
```

```
        //invio file(s)
```

```
        if(dir.isDirectory()) {
```

```
            //creazione Socket per dir corrente
```

```
            socket = new Socket(addr,serverPort);
```

```
            socketIn = new DataInputStream(socket.getInputStream());
```

```
            socketOut = new DataOutputStream(socket.getOutputStream());
```

```
            //ciclo nei file della directory
```

```
            for(File f : dir.listFiles())
```

```
                InvioFile(f, min_size, socketIn, socketOut);
```

```
            socketOut.writeUTF("finito");
```

```
            socket.shutdownOutput();
```

```
        }
```

```
    } else {
```

```
        System.out.println(read_dir + " non è una directory");
```

```
    }
```

```
    }catch(IOException e) {
```

```
        e.printStackTrace();
```

```
        System.exit(4);
```

```
    }
```

```
}while(read_dir!=null);
```

Controlliamo ad ogni ciclo che l'utente inserisca una directory valida da INPUT.

Invio TUTTI i files
(accettati)
all'interno della dir.
Comunico al server
la fine dei files.
Chiudo l'out della
Socket lato Client.
(NUOVA
CONNESSIONE
AD OGNI
DIRECTORY)

Creiamo la socket che
verrà utilizzata per
trasferire i file dalla dir al
Server.

L'utente può scegliere di inviare
altre directorys o terminare con
EOF. (readLine(); lo legge come
null).

Client(3): Metodo InvioFile

```
private static void InvioFile(File f, int min_size, DataInputStream socketIn, DataOutputStream socketOut) throws IOException {
```

```
    long file_size = f.length();
```

```
    String ans = null;
```

```
    if(file_size > min_size && !f.isDirectory()) {
```

Se il file da passare
rispetta la min_size e non
è una directory procediamo
alla richiesta.

```
        socketOut.writeUTF(f.getName());  
        ans = socketIn.readUTF();
```

```
        if(ans.equals("attiva")) {
```

```
            System.out.println("Sto inviando "+f.getName());
```

```
            socketOut.writeLong(file_size);
```

```
            FileUtility.trasferisci_a_byte_file_binario(new DataInputStream(new FileInputStream(f)), socketOut);
```

```
            System.out.println("Inviato");
```

```
        }
```

```
        else if (ans.equals("salta file")){
```

```
            System.out.println("File "+f.getName()+" già esistente o non richiesto dal server.");
```

```
        }
```

```
        else {
```

```
            System.err.println("Errore nel trasferimento del file "+f.getName()+" . Risposta dal server non prevista: "+ans);
```

```
            System.exit(1);
```

```
        }
```

```
    }
```

```
    if(f.isDirectory()) {
```

```
        for(File file: f.listFiles())
```

```
            InvioFile(file,min_size,socketIn,socketOut);
```

```
    }
```

```
}
```

Inviando il nome del file al Server che ci dirà se
Inviare o saltare il file tramite la socket creata in
precedenza.

Nel caso in cui il file sia in realtà una
subDirectory ricorsivamente inviamo tutti
i file presenti al suo interno.

Server(1): ServerSocket

```
if (args.length == 0) {  
    port = PORT;  
}  
else{
```

Se il Server viene avviato senza argomenti, la ServerSocket verrà inizializzata con una porta di default PORT.

```
    try {  
        port = Integer.parseInt(args[0]);  
    } catch (NumberFormatException e) {  
        System.err.println("usage: Server [porta(intero)] (DefaultPort: 1026)");  
        System.exit(1);  
    }  
}
```

Se invece passiamo come primo argomento la porta desiderata, la ServerSocket verrà avviata su questa porta. (A meno di errori di formato :-).

```
try {  
    serverSocket = new ServerSocket(port);  
} catch (IOException e) {  
    System.err.println("Errore creazione serverSocket");  
    e.printStackTrace();  
    System.exit(2);  
}
```

Server(2): Socket → Thread(socket)

```
while(true) {  
    try {  
        System.out.println("Server PRONTO in attesa di una richiesta da un Client.");  
        socket = serverSocket.accept();  
    } catch (IOException e) {  
        e.printStackTrace();  
  
        try {  
            serverSocket.close();  
        } catch (IOException e1) {  
            System.err.println("Errore chiusura serverSocket:");  
            e1.printStackTrace();  
            System.exit(2);  
        }  
        System.exit(2);  
    }  
    DirectoryServerThread thread = new DirectoryServerThread(socket);  
    thread.run();  
}
```

→ Crea la ServerSocket, il Server si dovrà occupare di attendere la richieste dai Clients ed assegnarle ai suoi Threads.

L'accept, funzione sospensiva della ServerSocket, si occupa della creazione della socket che verranno utilizzate dal Thread per comunicare con il Client corrispondente.

→ Infine il Server inizializza il DirectoryServerThread passandogli la socket e lo avvia (run()).

DirectoryServerThread(1): Ciclo ricezione file

```
public void run() {  
    boolean finito =false;  
  
    try {  
        while(!finito) {  
  
            nomeFile = inSock.readUTF();  
  
            if(nomeFile.equals("finito")) {  
                finito=true;  
                socket.shutdownOutput();  
                return;  
            }  
            else {  
                System.out.println("Thread:"+Thread.currentThread().getId()+" Richiesta invio FILE: "+nomeFile);  
            }  
  
            if(new File(nomeFile).exists()) {  
  
                outSock.writeUTF("salta file");  
                System.out.println("Thread:"+Thread.currentThread().getId()+" File GIA' presente");  
            }else {  
  
                outSock.writeUTF("attiva");  
  
                size = inSock.readLong();  
  
                FileUtility.trasferisci_a_byte_file_binario(inSock, new DataOutputStream(new FileOutputStream(nomeFile)),size);  
                System.out.println("Thread:"+Thread.currentThread().getId()+" File ricevuto");  
            }  
        }  
    }  
}
```


Il Client dopo aver trasferito tutti i file della Directory, notificherà al DirectoryServerThread la fine del trasferimento, che chiuderà l'out della Socket del Server e terminerà.

Se il file esiste, il Server dice al Client di saltarlo.

Se il file NON esiste, il Server dice al Client di attivare il trasferimento e si occupa di trasferire i byte sul file binario tramite il metodo `trasferisci_a_byte_file_binario`

DirectoryServerThread(1bis): Eccezioni

```
catch(IOException e) {  
  
    try {  
        socket.close();  
    } catch (IOException e1) {  
        System.err.println("Thread:"+Thread.currentThread().getId()+" Errore chiusura socket:");  
        e1.printStackTrace();  
        return;  
    }  
    System.err.println("Thread:"+Thread.currentThread().getId()+" Stream Chiuso. Problema:");  
    e.printStackTrace();  
    return;  
}
```

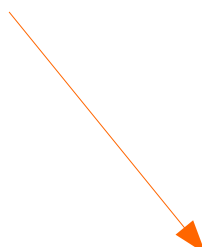


Nel caso ci siano eccezioni nell'esecuzione, il Thread si occupa di **chiudere la Socket** e **termina l'esecuzione**.

FileUtility:

Metodo con sizeFile

```
static protected void trasferisci_a_byte_file_binario (DataInputStream src, DataOutputStream dest, long size) throws IOException {  
    // ciclo di lettura da sorgente e scrittura su destinazione  
    int buffer = 0;  
  
    try  
    {  
        // esco dal ciclo alla lettura di un valore negativo ossia EOF o quando raggiungo la dimensione attesa del file (size)  
        for(int i=0; i<size && ((buffer = src.read()) >= 0); i++)  
            dest.write(buffer);  
  
        dest.flush();  
    }catch (EOFException e){  
        System.out.println("Problemi:");  
        e.printStackTrace();  
    }  
}
```



Il **metodo** differisce dal metodo originale perché si dovrà occupare di terminare il trasferimento del file quando sarà raggiunta la dimensione del file **size** che ci è stata inviata dal Client.