SOCKET JAVA CON CONNESSIONE

ESERCITAZIONE 2 RDC

26 OTTOBRE 2021

Bonantini Alessandro
Galeone Christian
Garcia Fernandez Diego
Piras Gabriele
Predieri Nicholas

FUNZIONAMENTO

❗ **N.B: UTILIZZARE LA STESSA SOCKET PER I TRASFERIMENTI**

PASSO 1

Il Client si connette al Server con una **connessione TCP**.
Il Client chiede all'utente il **nome del direttorio** da inviare al server e la **dimensione minima** dei file da accettare.

PASSO 2

Successivamente controllerà che **ogni file** del direttorio fornito rispetti la **dimensione minima**, invierà al Server il nome e attenderà l'esito.

PASSO 3

Il Server (progettato sia come parallelo che sequenziale) risponde al Client con **"attiva"** nel caso il file non sia già presente nel direttorio corrente; **"salta file"** altrimenti.

PASSO 4

Il Client invia al Server la dimensione e il file accettato tramite **stream di byte**. Il file verrà salvato nel direttorio del Server. Terminato l'invio del direttorio, il Client ripeterà la procedura fin quando l'utente non inserisce **EOF**.

CODICE

Let’s start with the code

Client.java

```
// Ciclo for per controllare i file all'interno della directory
                for (File f : new File(nomeFile).listFiles()) {
    // se il file è un direttorio, passo al file successivo
                    if (f.isDirectory()) {
                        System.out.println("File " + f.toString() + " è una directory");
                        continue;
                    }
    // se il file dimensione minore della minima richiesta, passo al successivo
                    if (f.length() < sizeMin) {
                        System.out.println("File " + f.toString() + " al di sotto della
soglia " + sizeMin + " bytes" + " [" + f.length() + " bytes]");
                        continue;
                        }
```

Client.java

```
String esito;
                try{
                        esito = inSock.readUTF();
                        System.out.println("Esito trasmissione: " + esito);
                        if (esito.equalsIgnoreCase("Salta File")) {
                          continue;
                         } else { // Esito: Attiva
                           /* Trasferimento file */
                           // creazione stream di input da file
                                try{
                                    inFile = new FileInputStream(f);
```

Client.java

```
try {
        outSock.writeUTF(String.valueOf(f.length()));
        System.out.println("Inviato la dimensione del file " + f.getName());
              } catch (Exception e) {
                   System.out.println("Problemi nell'invio della dimensione di " + f
                   .getName()+ ": ");
                    e.printStackTrace();
                    System.exit(-2);
                 }
                // trasferimento file
       try{
              DataInputStream fDataInputStream = new DataInputStream(inFile);
              outSock.write(fDataInputStream.readAllBytes(), 0, (int)f.length());
              inFile.close();              // chiusura file
             System.out.println("Trasmissione di " + nomeFile + " terminata ");  }
```

SequentialServer.java

```
//  Finche' il cliente e' connesso
                while (clientSocket.isConnected()) {
                    nomeFile = inSock.readUTF();
                    //elaborazione e comunicazione esito
                    FileOutputStream outFile = null;

                    if (nomeFile == null) {
                        System.out.println("Problemi nella ricezione del nome del
file: ");
                        clientSocket.close();
                        continue;
```

SequentialServer.java

```
        else {
                                System.out.println("Ho ricevuto il nomeFile: " + nomeFile);
                                boolean found = false;
                                // controllo su file
                                for (File f : wDir.listFiles()) {
                                    if(nomeFile.equals(f.getName())){
                                        System.out.println("Il file " + nomeFile + " e' già
presente nel direttorio");
                                        found = true;
                                        break;
}
```

SequentialServer.java

```
    if(found){
          outSock.writeUTF("Salta File");
    }else{
          System.out.println("Il file " + nomeFile + " non e' presente nel
direttorio");
          outSock.writeUTF("Attiva");
           }
```

SequentialServer.java

```
   // ricezione dimensione
            sizeFile = Long.parseLong(inSock.readUTF());
            System.out.println("Ricevo il file " + nomeFile + ": [" + sizeFile + "
bytes] \n");
            byte[] rBytes = new byte[(int)sizeFile];
            inSock.read(rBytes, 0, (int)sizeFile);
            outFile.write(rBytes);
```

ParallelServer.java

```
// servizio delegato ad un nuovo thread
                try {
                    new ServerConThread(clientSocket, wDir).start();
                }
catch (Exception e) {
                    System.err.println("Server: problemi nel server thread: "
                            + e.getMessage());
                    e.printStackTrace();
                    continue;
                }
```

ParallelServer.java

```
class ServerConThread extends Thread{
   private Socket clientSocket = null;
   private File directory = null;

   public ServerConThread(Socket clientSocket, File directory) {
       this.clientSocket = clientSocket;
       this.directory = directory;
   }
public void run()
```

CONCLUSIONI

I punti focali nell'esercitazione sono i seguenti:

- Il passaggio di parametri con tipi distinti(interi e file) attraverso lo stream di byte
- Gestione e controllo di file di tipo direttorio
- Confrontare i punti di forza e le criticità di server sequenziali e concorrenti

FINE