ESERCITAZIONE 2

Socket in Java con connessione

DAVIDE DI MOLFETTA

MIRKO LEGNINI

DANIELE NANNI CIRULLI

NATANAELE STAGNI

LORENZO VENERANDI

OBIETTIVO

L’obiettivo dell’esercitazione è sviluppare un protocollo di trasferimento file basato sul socket TCP.

REQUISITI DI PROGETTO

Il progetto richiede di stabilire una connessione TCP tra Cliente e Servitore per trasferire i File contenuti in una Directory.

Il Cliente specifica il nome della Directory e la dimensione minima dei File tramite uno scambio di messaggi sulla connessione, poi procede con i trasferimenti.

È richiesta una implementazione sia parallela che sequenziale per il Servitore.

MULTIPLE PUT CLIENT

OBIETTIVI:

- Stabilire una connessione con il server tramite socket TCP.
- Comunicare al server la directory desiderata ed attendere conferma.
- Controllare che la dimensione dei file all'interno di essa sia maggiore della soglia desiderata.
- Inviare al server il nome dei files e il loro contenuto ed attendere l'esito dell'operazione.
- Chiudere la socket una volta terminati tutti i trasferimenti.

MULTIPLE PUT CLIENT

```
BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
System.out.print(
        "MultiplePutClient Started.\n\n^D(Unix)/^Z(Win)+invio per uscire, oppure immetti nome directory: ");

try {
    while ((nomeDir = stdIn.readLine()) != null) {

        // creazione socket
        try {
            socket = new Socket(addr, port);
            System.out.println("Creata la socket: " + socket);
        } catch (Exception e) {
            System.out.println("Problemi nella creazione della socket: ");
            e.printStackTrace();
        }
```

```
// creazione stream di input/output su socket
try {
    inSock = new DataInputStream(socket.getInputStream());
    outSock = new DataOutputStream(socket.getOutputStream());
} catch (IOException e) {
    System.out.println("Problemi nella creazione degli stream su socket: ");
    e.printStackTrace();
}
```

MULTIPLE PUT CLIENT

```
if (new File(nomeDir).isDirectory()) { // Il file passato è una directory

    System.out.print("\nImmetti dimensione minima file: ");
    try {
        minFileSize = Integer.parseInt(stdIn.readLine().trim());
    } catch (NumberFormatException e) {
        System.out.println("Dimensione file errata!");
        continue;
    }

} else {
    // Directory non trovata o file
    System.out.println("Directory non identificata");
    System.out.print("\n^D(Unix)/^Z(Win)+invio per uscire, oppure immetti nome file: ");
    continue;
}
```

```
File[] filesArray = new File(nomeDir).listFiles();
if (filesArray == null) {
    System.out.println("Nella directory non sono presenti files");
}

outSock.writeUTF(nomeDir);
if (!inSock.readUTF().equals("conferma")) {
    System.out.println("Conferma directory non ricevuta");
    continue;
}
```

MULTIPLE PUT CLIENT

```
for (File f : filesArray) {
    if (f.isFile() && f.length() >= minFileSize) { // Check dimensioni del file

        outSock.writeUTF(f.getName());
        System.out.println("\n\nInviato il nome del file " + f.getName());

        if (inSock.readUTF().equals("attiva")) {

            System.out.println("Inizio la trasmissione di " + f.getName());
            inFile = new FileInputStream(f);
            FileUtility.trasferisci_a_byte_file_binario(new DataInputStream(inFile), outSock);
            inFile.close(); // chiusura file

            System.out.println("Trasmissione di " + f.getName() + " terminata ");
            System.out.println(
                    "Esito trasmissione: " + inSock.readUTF() + "\n--------------------\n");

        } else {
            System.out.println(f.getName() + " non sarà inviato");
        }

    } else {
        if (f.isFile())
            System.out.println(
                    "Il file " + f.getName() + " non raggiunge la dimensione minima selezionata");
    }
}
```

MULTIPLE PUT SERVER

OBIETTIVI:

- Mettersi in ascolto sulla porta indicata.
- Stabilire una connessione con i clienti su richiesta.
- Ricevere il nome della Directory da cui copiare i File.
- Ricevere i nomi dei singoli File e copiare il contenuto di quelli non preesistenti.
- Restituire l'esito di ciascuna operazione.

MULTIPLE PUT SERVER

```
while (true) {
    System.out.println("Server: in attesa di richieste...\n");

    try {
        // bloccante fino ad una pervenuta connessione
        clientSocket = serverSocket.accept();
        // clientSocket.setSoTimeout(30000);
        System.out.println("Server: connessione accettata: " + clientSocket);
    } catch (Exception e) {
        System.err.println("Server: problemi nella accettazione della connessione: " + e.getMessage());
        e.printStackTrace();
        continue;
    }
```

MULTIPLE PUT SERVER

```
outSock.writeUTF("conferma");
String nomeFile;
FileOutputStream outFile = null;
File curFile = null;

File directory = new File(dir);
directory.mkdir();
```

```
ile ((nomeFile = inSock.readUTF()) != null) {

    curFile = new File(dir + "/" + nomeFile);
    System.out.println(nomeFile);

    if (curFile.exists())
        outSock.writeUTF("salta");
    else {
        curFile.createNewFile();
        outSock.writeUTF("attiva");

        // ciclo di ricezione dal client, salvataggio file e stamapa a video
        try {
            outFile = new FileOutputStream(curFile);

            System.out.println("Ricevo il file " + nomeFile + ": \n");
            FileUtility.trasferisci_a_byte_file_binario(inSock, new DataOutputStream(outFile));
            System.out.println("\nRicezione del file " + nomeFile + " terminata\n");

            outFile.close();
            outSock.writeUTF("conferma");
            outSock.flush();
```

SEQUENZIALE VS PARALLELO

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

L'algoritmo del server parallelo è analogo a quello del sequenziale, ma viene delegato ad un Thread.

```
public MultiPutServerThread(Socket clientSocket) {
    this.clientSocket = clientSocket;
}

public void run() {
```

CONCLUSIONI

- Il Client è agnostico rispetto alle caratteristiche del Server, possiamo usare Server sequenziale e parallelo indifferentemente.
- Il Server parallelo risulta più efficiente a causa del minor tempo di accodamento.