



ESERCITAZIONE 2

Socket in Java con connessione

DAVIDE DI MOLFETTA

MIRKO LEGNINI

DANIELE NANNI CIRULLI

NATANAELE STAGNI

LORENZO VENERANDI

A decorative graphic on the left side of the slide, consisting of a network of light blue lines and small circles, resembling a circuit board or a neural network, extending from the top and bottom edges.

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");
    }
}
```

```
while ((buffer=src.read()) > 0) {
    dest.write(buffer);
}
dest.write('\0');
dest.flush();
```



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();
```

```
while ((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 stampa 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

```
// servizio 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() {
```


ESECUZIONE CLIENT - SERVER

The screenshot displays the Visual Studio Code IDE with two Java files open: `MultiplePutClient.java` and `MultiPutServer.java`. The Client file is on the left, and the Server file is on the right. The terminal at the bottom shows the execution of both programs.

```
Client > src > MultiplePutClient.java > MultiplePutClient > main(String[])
1  import java.net.*;
2  import java.io.*;
3
4  public class MultiplePutClient {
5
6      Run | Debug
7      public static void main(String[] args) throws IOException {
8
9          InetAddress addr = null;
10         int port = -1;
11
12         try { // Controllo argomenti
13             if (args.length == 2) {
14                 addr = InetAddress.getByName(args[0]);
15                 port = Integer.parseInt(args[1]);
16             } else {
17                 System.out.println("Usage: java MultiplePutClient
18                 System.exit(1);
19             }
20         } catch (Exception e) {
21
22         }
23     }
24 }
```

```
Server > src > MultiPutServer.java > MultiPutServer > main(String[])
1
2  import java.io.IOException;
3  import java.net.*;
4
5  public class MultiPutServer {
6
7      public static final int PORT = 1050; // default port
8
9      Run | Debug
10     public static void main(String[] args) throws IOException {
11
12         int port = -1;
13
14         try { // Controllo argomenti
15             if (args.length == 1) {
16                 port = Integer.parseInt(args[0]);
17                 if (port < 1024 || port > 65535) {
18                     System.out.println("Usage: java LineServer [s
19                     System.exit(1);
20                 }
21             }
22         }
23     }
24 }
```

Terminal Output:

```
PS C:\Users\lollo\Documents\Coding\Reti-di-Calcolatori-gruppo-Lore\Eserci
c> cd 'c:\Users\lollo\Documents\Coding\Reti-di-Calcolatori-gruppo-Lore\Eser
citazione_2\Client'; & 'c:\Users\lollo\vscode\extensions\vscjava.vscode-jav
a-debug-0.36.0\scripts\launcher.bat' 'C:\Program Files\Java\jdk-17.0.1\bin\j
ava.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=loca
lhost:62460' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-
Dfile.encoding=UTF-8' '-cp' 'C:\Users\lollo\Documents\Coding\Reti-di-Calcola
tori-gruppo-Lore\Esercitazione_2\Client\bin' 'MultiplePutClient' 'localhost'
'1050'
MultiplePutClient Started.

^D(Unix)/^Z(Win)+invio per uscire, oppure immetti nome directory: []

Server: in attesa di richieste...

PS C:\Users\lollo\Documents\Coding\Reti-di-Calcolatori-gruppo-Lore\Esercita
zione_2\Client> c:; cd 'c:\Users\lollo\Documents\Coding\Reti-di-Calcolatori
-gruppo-Lore\Esercitazione_2\Server'; & 'c:\Users\lollo\vscode\extensions
\vscjava.vscode-java-debug-0.36.0\scripts\launcher.bat' 'C:\Program Files\J
ava\jdk-17.0.1\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,s
uspend=y,address=localhost:62453' '--enable-preview' '-XX:+ShowCodeDetailsI
nExceptionMessages' '-Dfile.encoding=UTF-8' '-cp' 'C:\Users\lollo\Documents
\Coding\Reti-di-Calcolatori-gruppo-Lore\Esercitazione_2\Server\bin' 'MultiP
utServer'
MultiPutServerCon: avviato
Server: crea la server socket: ServerSocket[addr=0.0.0.0/0.0.0.0,localpor
t=1050]
Server: in attesa di richieste...

[]
```



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.
 - Ogni thread si occupa del trasferimento dell'intero direttorio scelto dal client, evitando quindi ulteriori operazioni di apertura/chiusura della socket.
- 