Gruppo 17

Università degli Studi di Bologna
Scuola di Ingegneria

Esercitazione 2

Socket Java con connessione

Corso di Reti di Calcolatori T

Daniele Magagnoli, Giulia Martina Bal,
Antonio Cassanelli, Gianmiriano Porrazzo

Anno Accademico 2021/2022

Specifiche dell'esercitazione:

- L'applicazione deve trasferire dal Client al Server tutti i file di un direttorio.
- Il Client chiede all'utente il nome del direttorio, assoluto oppure relativo, connettendosi al Server successivamente con una connessione (tramite una Socket) per inviare le informazioni (nome file, contenuto e dimensione).
- Il Server risponde «attiva» se un file con quel nome non è presente nel file system, risponde «salta file» altrimenti.
- Il Server attende una richiesta di connessione da parte dei clienti e usa la socket client per creare uno stream di input, da cui ricevere il nome e il contenuto del file, e uno stream di output per attivare il trasferimento.
- Il server deve essere realizzato sia come server sequenziale sia come server parallelo.

CLIENT

Dopo aver creato la socket con gli argomenti passati in ingresso e gli stream di input e output per comunicare con il server, il Client chiede all'utente di inserire la cartella.

Successivamente, faccio i controlli sulla cartella e chiedo la dimensione massima del file da trasmettere.

Per ogni file della cartella invio il nome al server tramite la socket stream, se il file non è presente nel file system del server, il client invierà la sua dimensione.

```
BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
System.out
    .print("Client Started.\n\n^D(Unix)/^Z(Win)+invio per uscire");

System.out.print("\n\nInserire directory:");

while((dir=stdIn.readLine())!=null) {
    cartella=new File(dir);
    int dim=-1;

    if(!cartella.exists() || !cartella.isDirectory() ){
        System.out.println("L'argomento passato non esiste o non una directory.");
        System.out.println("Inserisci una directory:(Crlt+D per terminare)");
        continue;
    }

    System.out.println("Inserisci la massima dimensione di file:");
    try {
        dim = Integer.parseInt(stdIn.readLine());
    }
    catch(NumberFormatException e){
        System.out.println("Inserisci un intero!");
        socket.close();
        System.exit(1);
    }
```

```
for(File f : cartella.listFiles()) {
    if(f.length()< dim)
        continue;
    else {
        // trasmissione del nome
        try{
            outSock.writeUTF(f.getName());
            System.out.println("Inviato il nome del file " + f.getName());
        }
        catch(Exception e){
            System.out.println("Problemi nell'invio del nome di " + f.getName()
                + ": ");
            e.printStackTrace();
            System.out
                .print("\n^D(Unix)/^Z(Win)+invio per uscire");
        // il client continua l'esecuzione riprendendo dall'inizio del ciclo
            continue;
        }

        // ricezione esito
        String esito;
        try{
            esito = inSock.readUTF();
            if(esito.equalsIgnoreCase("attiva")) {
                // trasmissione della dim del file
                try{
                    outSock.writeLong(f.length());
                    System.out.println("Inviata la dimensione del file " + f.getName());
```

CLIENT

Se il file non è presente, il server risponde con la stringa «attiva» e il client lo invia con la funzione presente nella classe FileUtility.

In caso contrario il server invia la stringa «salta file» e passa a quello successivo. Nel caso sia l'ultimo file, chiude la socket stream e chiede all'utente il nome di una nuova cartella.

```
    // trasferimento file
    try{
        System.out.println("Inizio la trasmissione di " + f.getName());
        inFile= new FileInputStream(f.getAbsolutePath());
        FileUtility.trasferisci_N_byte_file_binario(new DataInputStream(inFile), outSock, f.length());
        inFile.close();
        System.out.println("Trasmissione di " + f.getName() + " terminata\n ");
    }
    catch(SocketTimeoutException ste){
        System.out.println("Timeout trasmissione file scattato: ");
        ste.printStackTrace();
        socket.close();
        System.out
            .print("\n^D(Unix)/^Z(Win)+invio per uscire\n");
        // il client continua l'esecuzione riprendendo dall'inizio del ciclo
        continue;
    }
    catch(Exception e){
        System.out.println("\nProblemi nell'invio di " + f.getName() + ": ");
        e.printStackTrace();
        socket.close();
        System.out
        .print("\n^D(Unix)/^Z(Win)+invio per uscire");
        // il client continua l'esecuzione riprendendo dall'inizio del ciclo
        continue;
    }
}else if(esito.equalsIgnoreCase("salta file")) {
    System.out.println(f.getName() + " già esistente\n");
    continue;
}
else {
    System.out.println("Stringa esito non valida\n");
    continue;
}
```

FILE UTILITY

La funzione trasferisce byte in base alla dimensione del file.
Dopo aver trasferito l'ultimo, stampa a video il numero di byte trasferiti, in caso contrario lancia un'eccezione.

```
static protected void trasferisci_N_byte_file_binario(DataInputStream src,
        DataOutputStream dest, long daTrasferire) throws
    IOException{
    int cont=0; //numero di ciclo di lettura / scrittura
    int buffer=0;
    try{
        //esco se ho raggiunto il numero di byte da trasferire
        while(cont<daTrasferire){
            buffer=src.read();
            dest.write(buffer);
            cont++;
        }
        dest.flush();

        System.out.println("Byte trasferiti: "+cont+"\n");
    }catch(EOFException e){
        System.out.println("Problemi, i seguenti: ");
        e.printStackTrace();
    }
}
```

FILE UTILITY

L’algoritmo trasferisce 1kb alla volta. Se il file ha una dimensione minore viene trasferito byte per byte.

```
//ALGORITMO CHE TRASFERISCE 1KB ALLA VOLTA
//PIU VELOCE MA ANCHE PIU DISPENDIOSO IN TERMINI DI RISORSE
static protected void trasferisci_TOT_byte_file_binario(DataInputStream src,
        DataOutputStream dest, long daTrasferire) throws IOException{


    //se il file è più piccolo di un 1kb trasferisco byte per byte
    if((int)daTrasferire/1000 == 0)
        trasferisci_N_byte_file_binario(src, dest, daTrasferire);
    //se Ã¨ piu grande trasferisco un 1kb.
    else {
        //mi preparo l'array che immagazzina i dati
        byte[] trasfArray = new byte[1000];
        int i = 0;
        try{
            int nCicli = (int)daTrasferire/1000;

            while(i < nCicli){
                src.read(trasfArray, 0, 1000);
                dest.write(trasfArray, 0, 1000);
                i++;
            }
            dest.flush();

            //se sono rimasti dei byte richiamo la funzione
            if(daTrasferire%1000 != 0)
                trasferisci_TOT_byte_file_binario(src, dest, daTrasferire-(nCicli*1000));

      //  System.out.println("Byte trasferiti: "+cont+"\n");
        }catch(EOFException e){
            System.out.println("Problemi, i seguenti: ");
            e.printStackTrace();
        }
    }
```

```
try {
    while((nomeFile = inSock.readUTF())!=null) {
        String esito;
        File curFile = new File(nomeFile);
        // controllo su file
        if (curFile.exists()) {
            try {
                System.out.println(nomeFile + " già esistente\n");
            }
            catch (Exception e) {
                System.out.println("Problemi nella notifica di file esistente: ");
                e.printStackTrace();
                continue;
            }
        }
        else { //ricezione file
            try {
                long sizeF=inSock.readLong();//prendo l'intero
                FileOutputStream outFile = new FileOutputStream(nomeFile);
                System.out.println("Ricevo il file " + nomeFile + " di dimensione "+sizeF);

                FileUtility.trasferisci_N_byte_file_binario(inSock,
                        new DataOutputStream(outFile),sizeF);
                System.out.println("Ricezione del file " + nomeFile
                        + " terminata\n");
                outFile.close();
                outSock.flush();
            }
            catch(SocketTimeoutException ste){
                System.out.println("Timeout scattato: ");
                ste.printStackTrace();
                clientSocket.close();
                System.out
                    .print("\n^D(Unix)/^Z(Win)+invio per uscire, solo invio per continuare: ");
                continue;
            }
            catch (Exception e) {
                System.err
                    .println("\nProblemi durante la ricezione e scrittura del file: "
                            + e.getMessage());
                e.printStackTrace();
                clientSocket.close();
                System.out.println("Terminata connessione con " + clientSocket);
```

SERVERSEQ

Il server sequenziale, dopo aver creato la server socket con la porta passata in ingresso, verifica per ogni file della cartella, se esiste o meno nel file system e lo comunica al client attraverso gli stream.
Se il server non trova il file, riceve la dimensione dal client e lo salva con la funzione presente nella classe FileUtility.

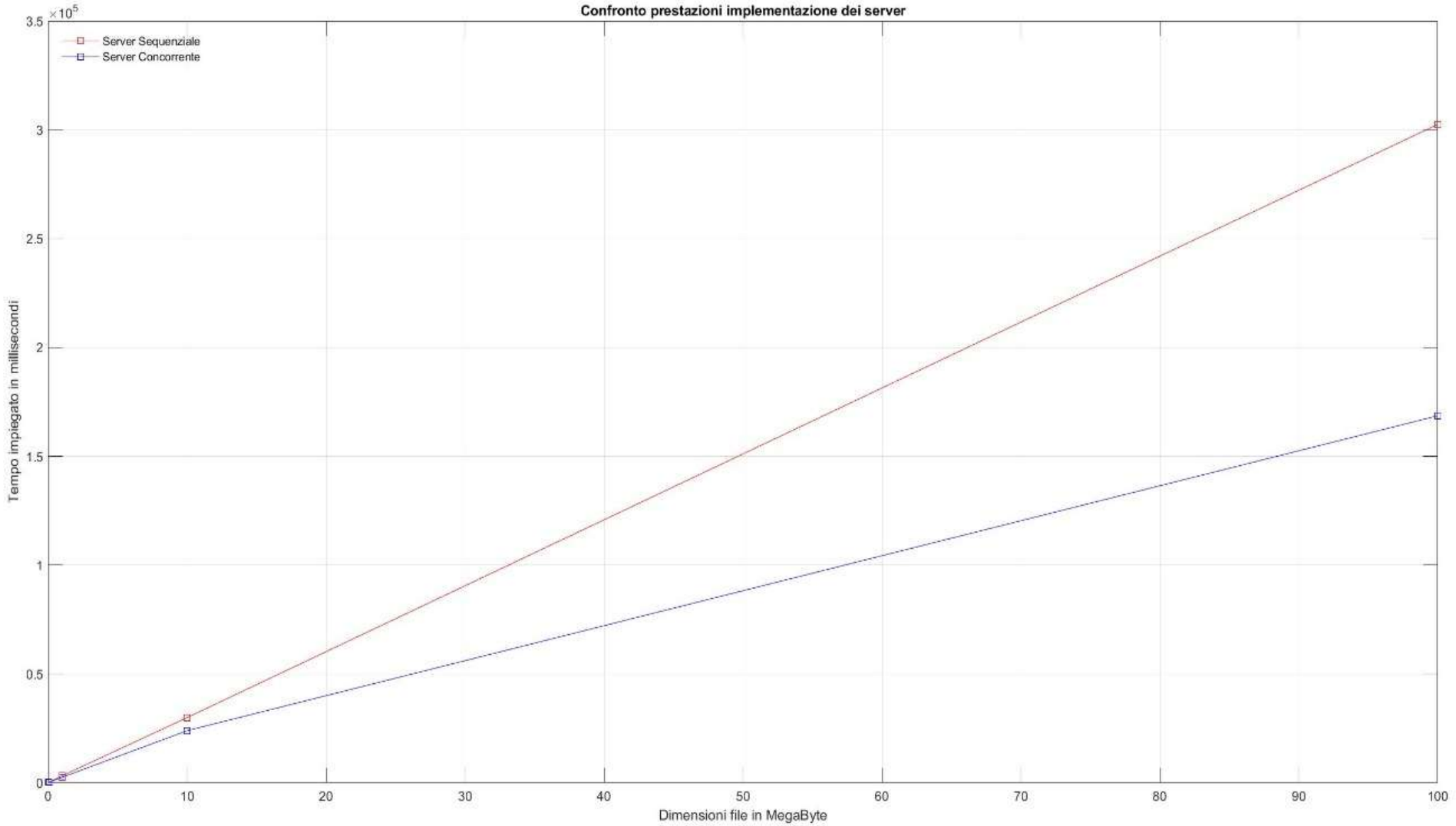
SERVERCON

Il server concorrente genera un nuovo processo figlio per ogni richiesta accettata da parte del client. Il processo figlio controlla la risposta del server e trasferisce il file binario se non è presente nel file system subito dopo aver ricevuto la sua dimensione.

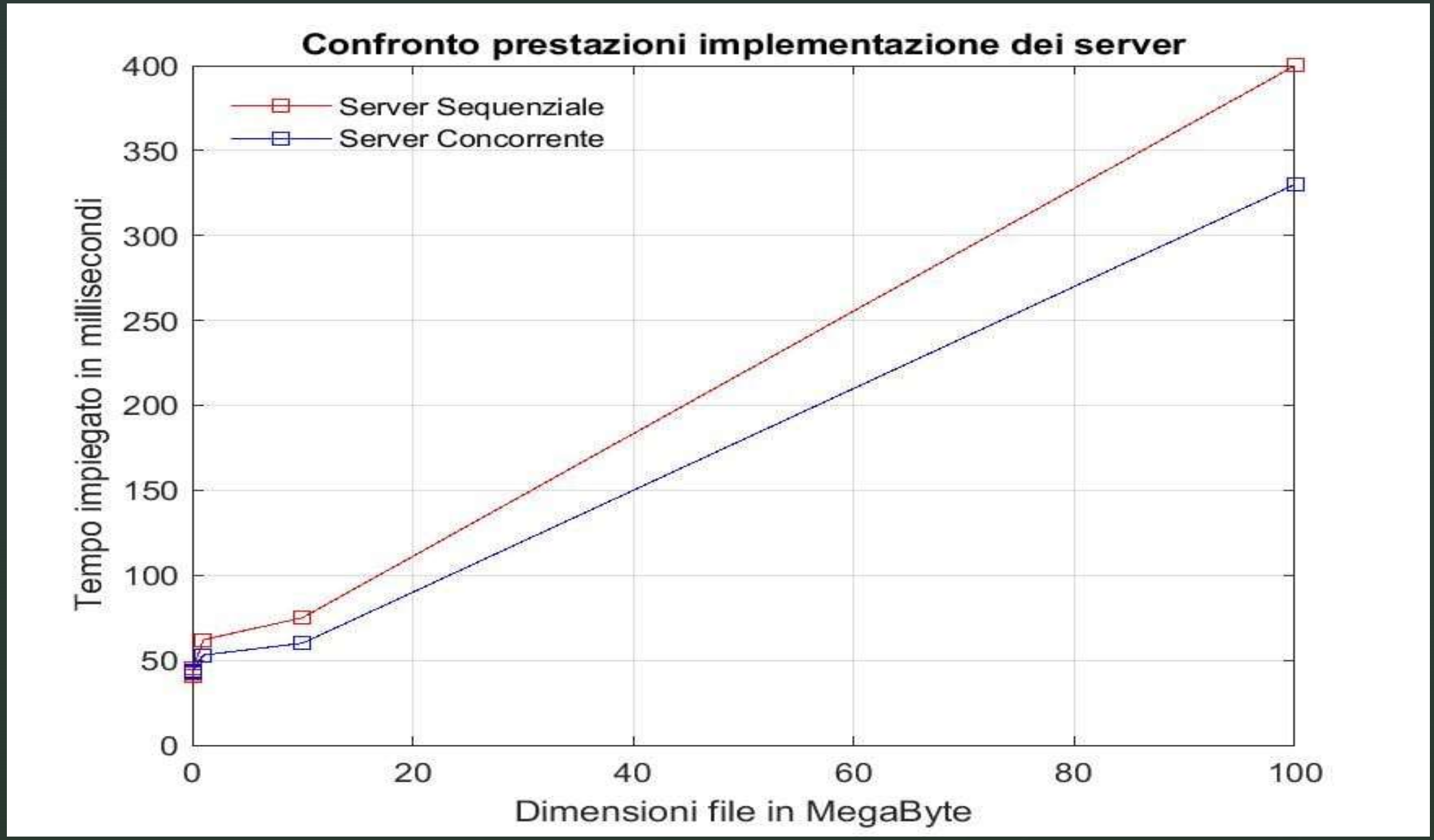
```
try {
    new ServerConThread(clientSocket).start();
}
catch (Exception e) {
    System.err.println("Server: problemi nel server thread: "
            + e.getMessage());
    e.printStackTrace();
    continue;
}
```

```
try {//prendere i file e vedere se esistono
    while((nomeFile = inSock.readUTF())!=null) {
        String esito;
        File curFile = new File(nomeFile);
        // controllo su file
        if (curFile.exists()) {
            try {

                System.out.println(nomeFile + " già esistente\n");
            }
            catch (Exception e) {
                System.out.println("Problemi nella notifica di file esistente: ");
                e.printStackTrace();
                continue;
            }
        }
        else {//ricezione file non presente nel file system
            try {
                long sizeF=inSock.readLong();//prendo l'intero
                outFile = new FileOutputStream(nomeFile);
                System.out.println("Ricevo il file " + nomeFile + " di dimensione "+sizeF);

                FileUtility.trasferisci_N_byte_file_binario(inSock,
                        new DataOutputStream(outFile),sizeF);
                System.out.println("Ricezione del file " + nomeFile
                        + " terminata\n");
                outFile.close();
```

FUNZIONE UTILIZZATA: trasferimento in base alla dimensione del file binario



FUNZIONE UTILIZZATA: trasferimento 1kB alla volta



GRAZIE PER L’ATTENZIONE