

# Esercitazione 2

Belano Andrea, Ceccolini Gabriele, Loddo Filippo, Merenda Simone.

# Specifiche del Progetto

Il Cliente si connette al Server con una connessione e la utilizza per lo scambio d'informazioni. Esso manda al Server i nomi dei file presenti sul directory specificato, il Server gli risponde con ATTIVA nel caso in cui il Server non possieda quel file nel suo directory, SALTA FILE se, invece, lo possiede.

# Cliente

Il Cliente si connette al Server, inviando, uno alla volta, i nomi dei file che superano una certa soglia di dimensione. Se il trasferimento viene accettato, invia anche la lunghezza, dopo di ché inizia il trasferimento che avviene in blocchi di 256KB. Una volta che tutti i file vengono elaborati, il cliente termina.

```
try {
    Socket socket = new Socket(InetAddress.getByName(host), port);
    out = new DataOutputStream(socket.getOutputStream());
    in = new DataInputStream(socket.getInputStream());
    socket.setSoTimeout(30000);
    for (File entry : dir.listFiles()) {

        if (entry.isFile() && (len = (int) entry.length()) >= minLen) {
            out.writeUTF(entry.getName());
            state = in.readInt();
            state = ATTIVA;
            if (state == ATTIVA) {
                out.writeInt(len);
                fileIn = new DataInputStream(new FileInputStream(entry));
                do {
                    read = fileIn.readNBytes(buffer, 0, BLOCK_SIZE);
                    out.write(buffer);
                } while (read == BLOCK_SIZE);
                fileIn.close();
            }
        }
    }

    socket.shutdownOutput();
    socket.shutdownInput();
    socket.close();

} catch (SocketTimeoutException e) {
    System.out.println("Timeout scaduto");
} catch (IOException e) {
    System.out.println("Errore di I/O");
    e.printStackTrace();
}
```

# FrontServer

Il FrontServer crea una ServerSocket e si mette in ascolto sulla porta 8000. Dopo aver accettato la richiesta di connessione, delega la gestione a un nuovo Thread passando come argomenti la ClientSocket e la directory dove avviene l'archiviazione dei file.

```
try {
    serverSocket = new ServerSocket(8000);
    serverSocket.setReuseAddress(true);
    while (true) {
        clientSocket = serverSocket.accept();
        clientSocket.setSoTimeout(30000);
        loadServer = new LoadServer(clientSocket, dir);
        loadServer.start();
    }
} catch (IOException e) {
    System.out.println("Impossibile creare la socket");
    System.exit(0);
}
```

# LoadServer

Il LoadServer prova a creare un nuovo file con il nome passato dal cliente. Se la creazione fallisce, vuol dire che esiste già; se ha successo, invia al cliente il segnale ATTIVA (1). In risposta il cliente invia la lunghezza del file e inizia il trasferimento del file in blocchi da 256KB. I blocchi vengono scritti sul file appena creato man mano che vengono ricevuti. Il file viene in fine chiuso e salvato. Questo procedimento viene ripetuto per ogni file presente nella directory del cliente.

```
private String UTFReader(DataInputStream reader) {
    try {
        return reader.readUTF();
    } catch (IOException e) {
        return null;
    }
}
```

```
try {
    byte block[] = new byte[BLOCK_SIZE];

    while ((fileName = UTFReader(reader)) != null) {

        file = Paths.get(dir.toString() + "\\" + fileName).toFile();

        if (file.createNewFile()) {

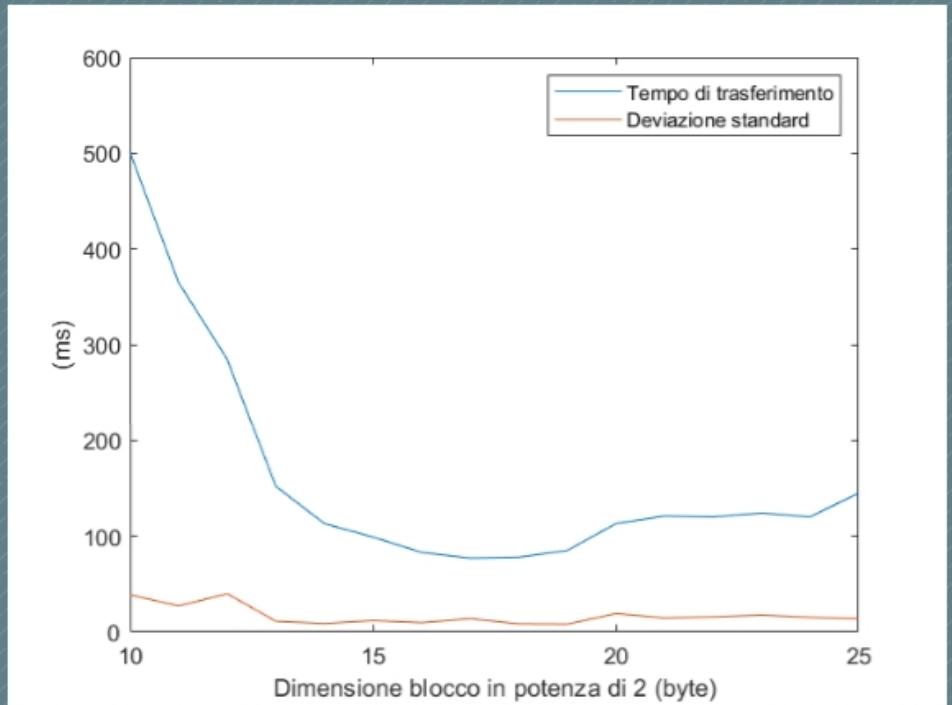
            writer.writeInt(ATTIVA);
            size = reader.readInt();
            fileWriter = new FileOutputStream(file);
            letti = 0;

            while (letti < size) {
                if (size - letti >= BLOCK_SIZE) {
                    lettiTmp = reader.readNBytes(block, 0, BLOCK_SIZE);
                    letti += lettiTmp;
                } else {
                    lettiTmp = reader.readNBytes(block, 0, size - letti);
                    letti += lettiTmp;
                }
                fileWriter.write(block, 0, lettiTmp);
            }
            fileWriter.close();
            writer.flush();
            output.flush();

        } else {
            writer.writeInt(SALTA);
        }
    }
    clientSocket.shutdownInput();
    clientSocket.shutdownOutput();
    clientSocket.close();
} catch (SocketTimeoutException e) {
    System.out.println("Timeout scaduto");
} catch (IOException e) {
    System.out.println("Errore di I/O");
    e.printStackTrace();
}
```

# Scelta della dimensione del blocco

Per aumentare l'efficienza, abbiamo svolto vari test su un file di 64MB per capire quale fosse la dimensione ottimale del blocco di trasferimento. Abbiamo dedotto che la dimensione ottimale è di 256KB.



# Grazie per l'ascolto

Belano Andrea, Ceccolini Gabriele, Loddo Filippo, Merenda Simone.