



**Università degli Studi di Bologna  
Scuola di Ingegneria**

# **Corso di Reti di Calcolatori T**

***Esercitazione 2 (proposta)  
Multiple PUT  
Socket Java con connessione***

**Antonio Corradi, Luca Foschini**

**Michele Solimando, Giuseppe Martuscelli, Marco Torello**

**Anno accademico 2020/2021**

# SPECIFICA CLIENT

---

Sviluppare un'applicazione C/S che effettui il **trasferimento dal client al server di tutti i file di un direttorio, la cui dimensione risulta maggiore di una soglia specificata dall'utente** (trasferimento **multiple put** con una condizione su **file size**). Il **Multiple put** viene file per file con assenso del server per ogni file.

Il client chiede all'utente il **nome del direttorio** (assoluto o relativo al direttorio corrente dove viene lanciato il cliente), si connette al server con una connessione (**java.net.Socket**), usandola per tutto il direttorio: i due versi dello stream sono usati, di output per **inviare tutte le informazioni** (nome, contenuto e dimensione), e di input per **ricevere il comando di attivazione del trasferimento per ciascun file**.

Il server fornisce **“attiva”** nel caso un file con quel nome non sia già presente sul file system del server, esito negativo altrimenti (ad esempio **“salta file”**). I file richiesti vengono dal server **salvati nel direttorio corrente** e il protocollo indicato evita che vengano sovrascritti file esistenti che abbiano lo stesso nome.

# SPECIFICA SERVER

---

Il server attende una richiesta di connessione da parte dei clienti (sulla server socket di ascolto `java.net.ServerSocket`), usa la socket client (`java.net.Socket`) connessa per creare uno stream di input **da cui ricevere i nomi dei file e il contenuto dei file**, e uno stream di output su cui **inviare il comando di attivazione trasferimento**.

Si noti che si **deve utilizzare la stessa connessione e socket per il trasferimento di tutti i file del direttorio**.

Il server deve essere realizzato come **server concorrente e parallelo**. Per ogni nuova richiesta ricevuta il **processo padre**, dopo aver accettato la richiesta, attiva un processo figlio a cui affida il completamento del servizio richiesto.

# NOTE PER LA REALIZZAZIONE

---

Per il trasferimento dei file bisogna studiare un **protocollo per la gestione del trasferimento multiplo**.

Ad esempio il client, **prima dell'invio del singolo file**, può inviare il **nome del file** e il **numero di byte** da cui il file è composto, quindi **invia lo stream di byte** se è il caso. Il server quindi si aspetta, per ogni file, il nome e il numero di byte, utilizzando **la medesima socket** per gestire tutti i trasferimenti.

Il ciclo si ripete fino a quando tutti i file nel direttorio richiesto sono stati inviati, quindi il client notifica la fine delle trasmissioni chiudendo la comunicazione e la connessione.

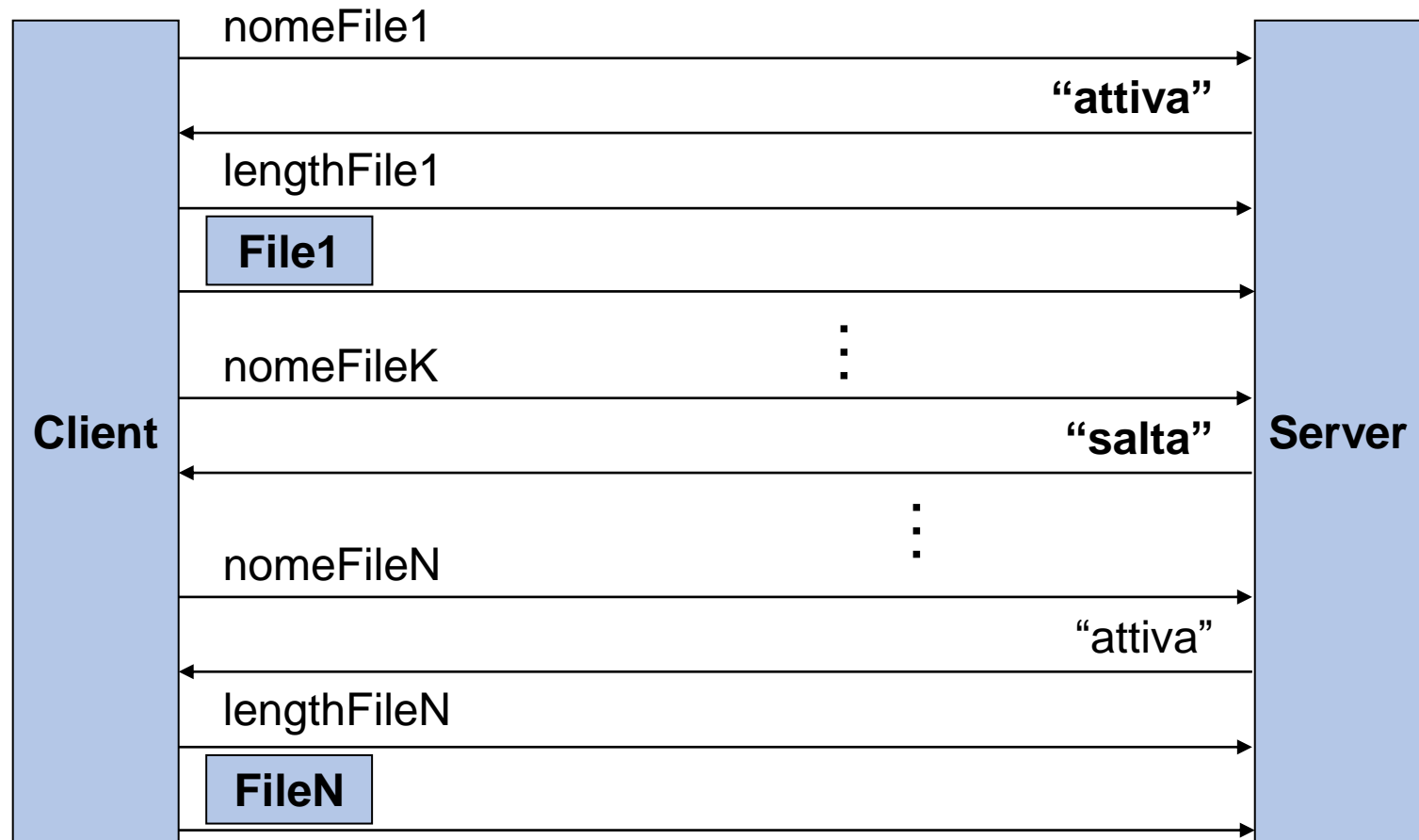
**La fine della connessione  
(chiusura socket )  
notifica la fine delle  
operazioni**



# NOTE PER LA REALIZZAZIONE

---

Più in dettaglio, si vuole realizzare il seguente protocollo:



# TRASFERIMENTO DI PIÙ DIRETTORI

(VERSIONE SEMPLIFICATA, PIÙ CONNESSIONI, UNA PER OGNI DIRETTORIO)

---

Per il trasferimento di più direttori bisogna studiare un opportuno **protocollo**. Anzitutto si sottolinea che il direttorio è una struttura locale che **non può** essere trasferita come un qualsiasi altro file dati.

Se **non ci interessa ricostruire la struttura in direttori (lato server)** è possibile riutilizzare il protocollo proposto in precedenza procedendo, per i vari direttori e i file all'interno di ciascuno di essi come segue: **prima dell'invio del singolo file, la cui dimensione risulta maggiore di un valore soglia**, il client può inviare il **nome del file** e il **numero di byte** da cui il file è composto, e successivamente **inviare lo stream di byte**. Il server segue questo protocollo per ogni file e utilizza **la medesima connessione** per gestire tutte le comunicazioni e salva tutti i file nel medesimo e unico direttorio corrente.

Il ciclo si ripete fino a che l'utente non indica l'intenzione di interrompere la sessione con un **fine file sulla connessione**, che provoca anche la chiusura della comunicazione del client con il server.



## PROPOSTA DI ESTENSIONE (PRELIMINARE): TRASFERIMENTO DIRETTORI (MGET)

---



Si estenda il programma sviluppato in modo che **gestisca il trasferimento di più direttori dal server al client** (multiple get di più direttori).

**Protocollo:** si estenda il protocollo in modo da abilitare il trasferimento di **diversi direttori** utilizzando la **stessa unica connessione dal cliente al server**.

Per ogni richiesta, **il Client** dovrà creare in locale un direttorio con lo stesso nome di quello richiesto, dove verranno salvati i file inviati dal server, quindi dovrà salvare in tale direttorio i file in arrivo dal server, e mettersi in attesa di una nuova richiesta dell'utente.



## PROPOSTA DI ESTENSIONE (PRELIMINARE): TRASFERIMENTO DIRETTORI (MGET)

---



**Per ogni richiesta di direttorio, il Server** deve inviare tutti i file del direttorio, e **notificare la fine della trasmissione** del direttorio stesso.

Si gestisca inoltre la **terminazione dell'interazione fra client e server**: il client **deve poter indicare al server** la propria intenzione di **chiusura dell'interazione**.

Una volta terminata la sessione client e server (processo figlio del server principale) terminano la propria esecuzione.





## PROPOSTA DI ESTENSIONE (DA CONSEGNARE): TRASFERIMENTO DIRETTORI (MGET/MPUT)

---



Si estenda ulteriormente il programma sviluppato in modo da abilitare il funzionamento del cliente servitore **in modalità sia get** (dal server al client) **che put** (dal client al server). Si studi in particolare come determinare i direttori di partenza e arrivo

**Protocollo:** si estendano i protocolli proposti, in modo da gestire la **sessione di lavoro fra client e server** utilizzando sempre la **stessa connessione**.

Per ogni richiesta, **il Client** richiede all'utente il tipo della richiesta che viene inoltrata al server (**mput** o **mget**); poi, **Client e Server** si coordinano per portare a termine l'operazione richiesta. Al termine, il client si pone in attesa di una nuova richiesta dell'utente (**mput** e **mget**) fino alla terminazione dell'interazione.