Relazione di progetto di PROGRAMMAZIONE DI RETI Anno 2021-2022

Cognome e Nome: Montanari Nicola
Matricola: 0000970119
Traccia: Traccia N.2
GitHub Repository: https://github.com/NIK4658/File-transfer-UDP

Architettura Client-Server UDP per trasferimento file

Scopo Progetto:

Lo scopo del progetto è quello di progettare e implementare in linguaggio Python un'applicazione Client-Server per il trasferimento di file che impieghi il servizio di rete senza connessione, utilizzando il protocollo User Datagram Protocol (UDP).

Struttura progettuale

Scelte generali:

Affinché i due processi (Client e Server) possano comunicare tra di loro, ho scelto di utilizzare un sistema di pacchetti di stringhe. Il valore di queste stringhe andrà a variare in base alla casistica del messaggio, errore oppure comando che dovrà essere interscambiato tra i due elementi.

(Un esempio può essere la richiesta di download di un determinato file all'interno del Server, dopo che il Client ha correttamente inviato il messaggio "get <filename>" al Server, quest'ultimo invierà una stringa al Client contenente "exist" nel caso che il determinato file esista veramente, oppure "error" nel caso esso non sia presente.)

Struttura Server:

Il Server ha la caratteristica di essere strutturato utilizzando un ciclo infinito nell'attesa di ricevere un qualsiasi comando da parte di un qualsiasi Client. Nel momento della ricezione di un messaggio contenente un comando, il Server si presterà a creare un Thread che avrà lo scopo di soddisfare la richiesta di tale comando appena ricevuto. Questo da garantire il continuo funzionamento del Server, anche in presenza di una molteplicità di Client che eseguono richieste nello stesso momento.

Operazioni preventive:

Nel momento in cui sia necessaria una trasmissione di file, verranno effettuate delle verifiche preventive per assicurarsi che tutto possa andare a buon fine senza incorrere in alcun tipo di problema. Utilizzerò spesso i termini “Mittente” e “Destinatario” piuttosto che “Client” e “Server” siccome entrambi i processi sono abilitati a fare sia Download che Upload e siccome il loro comportamento in certi aspetti è molto simile. Nel momento in cui viene effettuata una richiesta di Download o Upload, il mittente inizierà il processo di invio con una verifica di effettiva esistenza di quel determinato file nel percorso prestabilito (in questo caso il percorso coincide con il file del processo mittente). Nel caso il Client risulti essere il Destinatario e il Server il Mittente, dopo aver verificato l’esistenza o meno del file lato Server, quest’ultimo invierà un messaggio di responso al Client, che agirà di conseguenza.

Nel caso invece che il Client sia l’attuale Mittente, eseguirà una verifica di esistenza di tale file, per poi inviare la richiesta di UPLOAD al Server SOLO SE l’attuale verifica di esistenza risulti essere andata a buon fine.

Trasmissione di File:

Nel momento in cui la verifica di esistenza di un file risulta essere confermata, il mittente invierà per prima cosa il nome del file da trasferire, affinché il destinatario possa preventivamente creare un file omonimo vuoto dalla sua parte, pronto ad essere riempito dai dati che verranno trasmessi subito dopo.

Per il trasferimento di dati, lato mittente, ho optato di utilizzare un ciclo in cui ad ogni iterazione verrà letta e inviata una quantità di dati pari alla dimensione del pacchetto. Lato destinatario invece il file verrà aperto prima di entrare nel ciclo e verrà riempito di dati sempre a gruppi di grandezza pari alla dimensione del pacchetto.

La dimensione del pacchetto è definita dalla grandezza del buffer, settata di default pari a 1024 bit.

Lato mittente, il ciclo di trasmissione di dati viene concluso quando tutti i dati contenuti all'interno di un file sono stati correttamente inviati al destinatario.

Per il destinatario ho scelto di utilizzare un time-out per definire conclusa la comunicazione. Infatti se risultano essere passati più di 3 secondi di time-out dall'ultimo pacchetto ricevuto, significa che tutti i dati inviati dal destinatario sono arrivati a destinazione, il che significa che il file può essere ritenuto completato e con esso anche il processo di scambio di dati da entrambe le parti.

Singoli comandi nel dettaglio

Client:

- List: Esegue una richiesta di tipo “list” verso il Server e attende una sua risposta, che conterrà il numero di file all’interno della directory del Server, nel caso il numero sia 0, il Client visualizzerà a video che non sono presenti file in tale directory.
  Altrimenti il client abilita un time-out pari a 3 secondi e inizierà a ricevere i nomi dei file grazie ad una comunicazione tra il Client e il Server.

- Get <filename>: Esegue una richiesta di tipo “get <filename>” verso il Server e attende una sua risposta, che conterrà la stringa “exist” nel caso tale file richiesto esista, oppure “error” nel caso negativo.
  Il Client andrà ad abilitare un time-out pari a 3 secondi per poi iniziare a ricevere pacchetti pari a 1024 bit fino a completare il trasferimento di tale file richiesto. Una volta scaduto tale time-out, il processo di trasferimento verrà ritenuto concluso e verrà mostrato a video un messaggio per l’utente di avvenuto trasferimento.

- Put <filename>: Esegue una verifica di esistenza del file richiesto, e solo dopo aver confermato l’effettiva presenza di tale file nella sua directory, invia una richiesta di tipo “put <filename>” verso il Server.
  Il Client aprirà il file, per poi leggerlo e inviarlo al Server a cicli di 1024 bit fino al raggiungimento della fine del file. Una volta concluso il ciclo, il Client provvederà a chiudere il file e inviare un responso di corretto invio all’utente tramite la console.

Server:

- List: Alla ricezione del comando “list” da parte del Client, il Server come prima cosa invierà il numero di file presenti nella sua stessa directory (escludendo il file del server stesso). È presente un for-each che si occupa di inviare tutti i nomi dei file presenti al Client, che sarà già in attesa di ricevere tali pacchetti. Una volta inviati tali nomi, la richiesta sarà ritenuta conclusa.

- Get <filename>: Alla ricezione del comando “get <filename>” da parte del Client, il Server verificherà l’esistenza di tale file richiesto. Nel caso non esistesse, invierà un messaggio di errore al Client, viceversa, verrà inviato una stringa pari ad “exist”.
  Nel caso il file sia disponibile, il Server provvederà a aprire tale file, per poi leggerlo e inviarlo al Client utilizzando un ciclo while. Alla conclusione di tale ciclo, il trasferimento dati verrà ritenuto completato e verrà mostrato a video un messaggio di invio riuscito.

- Put <filename>: Alla ricezione del comando “put <filename>” da parte del Client, il Server provvederà a generare un file omonimo pronto ad essere riempito di dati e abiliterà un time-out pari a 3 secondi.
  Verrà poi eseguito l’effettivo trasferimento di dati con Client come mittente e Server come destinatario. Una volta scaduto tale time-out, il processo di trasferimento verrà ritenuto concluso e verrà mostrato a video un messaggio per l’utente di avvenuto trasferimento.

Funzionalità Software

Funzionalità Client:

Il Client è in grado di fornire le seguenti funzionalità con i relativi comandi:

- **List** : Richiede al Server una lista dei nomi dei file disponibili al download.
- **get <filename>** : Invia una Richiesta per il DOWNLOAD di un file specifico, che verrà eseguito immediatamente dopo aver ricevuto un messaggio di conferma da parte del Server. Oppure, ovviamente, il Client dovrà gestire l'errore in caso il server non sia in grado di inviare il file desiderato.
- **put <filename>** : Effettua l'UPLOAD sul Server di un file specificato dall'utente, presente nella cartella "Client". Riceve poi un messaggio di risposta con l'esito dell'operazione.

Funzionalità Server:

Il Server è in grado di fornire le seguenti funzionalità:

- Invio del messaggio di risposta al Client che ha precedentemente eseguito il comando "list". Il messaggio di risposta conterrà la **lista dei nomi dei file** presenti nella cartella "Server".
- Invio del messaggio di risposta al Client che ha precedentemente eseguito il comando "get <filename>". Il messaggio di risposta conterrà **il file richiesto**, se presente, od un opportuno **messaggio di errore.**
- Invio del messaggio di risposta al Client che ha precedentemente eseguito il comando "put <filename>". Il messaggio di risposta sarà **l'esito dell'operazione** di UPLOAD da parte del Client verso il Server.

Schema UML del dialogo tra Client e Server