

Relazione progetto reti
“Architettura client-server UDP per
trasferimento file”

Lorenzo Tosi
0000997569
lorenzo.tosi10@studio.unibo.it

27 luglio 2022

Indice

1	Obbiettivo	2
1.1	Requisiti	2
2	Progettazione	4
2.1	Scambio di file	4
2.2	Client e Server	6
2.2.1	Limitazioni	7
2.3	Timeout	7
3	Guida utente	8

Capitolo 1

Obbiettivo

Il progetto si pone come obbiettivo la realizzazione di un client-server UDP per il trasferimento dei file.

1.1 Requisiti

Il software deve permettere:

- Connessione client server senza autenticazione.
- La visualizzazione sul client dei file disponibili sul server.
- Il download di un file dal server.
- L'upload di un file sul server.

Funzionalità del client

- L'invio del messaggio List per richiedere la lista dei nomi dei file disponibili.
- L'invio del messaggio get per ottenere un file.
- La ricezione di un file richiesta tramite il messaggio di get o la gestione dell'eventuale errore.
- L'invio del messaggio put per effettuare l'upload di un file sul server e la ricezione del messaggio di risposta con l'esito dell'operazione.

Funzionalità del server

- Invio del messaggio di risposta al comando List al client richiedente.
- Il messaggio di risposta contiene la file List, ovvero la lista dei nomi dei file disponibili per la condivisione.
- L'invio del messaggio di risposta al comando get contenente il file richiesto, se presente, od un opportuno messaggio di errore.
- La ricezione di un messaggio put contenente il file da caricare sul server e l'invio di un messaggio di risposta con l'esito dell'operazione.

Capitolo 2

Progettazione

Per la costruzione del client e del server è stato utilizzato il linguaggio Python. Il client è rappresentato con il file `client.py`, mentre il server attraverso il file `server.py`. La comunicazione avviene attraverso lo scambio di messaggi che sono del tipo:

- Messaggi di comando: inviati dal client al server per richiedere l'esecuzione delle operazioni.
- Messaggi di risposta (Acknowledge): inviati dal server al client in risposta ad un messaggio di comando con l'esito dell'operazione.
- Dati: i dati che stanno venendo scambiati.

2.1 Scambio di file

Lo scambio di file Server-Client e viceversa è implementato in modo molto simile; entrambi svolgono queste azioni:

- Il sender calcola il numero nella quale il file verrà spezzettato, e il numero viene spedito al ricevitore.
- Il sender spaccetta il file e ogni pacchetto viene aggiunto ad una lista.
- Dopo aver spaccettato il file, il sender spedisce la lista. In dettaglio, spedisce ogni elemento contenuto nella lista, ovvero il file "spezzettato".
- Il ricevitore riceve i pacchetti che il sender ha inviato, e li inserisce in una lista.
- Dopo aver inviato tutti i pacchetti, il sender invia l'hash della lista al ricevitore.

- Il ricevitore calcola l'hash con la lista appena creata, e lo paragona con quello ricevuto.

La lista che le due parti si scambiano è formata da dati che presentano questa struttura:

- Al primo posto è presente la posizione ("pos"), ovvero l'indice che verrà utilizzato dal ricevitore per riordinare la lista di dati appena ricevuti in quanto non è garantito l'ordine.
- Al secondo posto è presente il valore ("value"), ovvero una parte dei bytes del file.

Il controllo che il file arrivato sia uguale al file spedito è garantito dalla funzione di hash, che prende la lista riordinata, calcola l'hash e ne ritorna una stringa. Se la stringa calcolata dal ricevente è uguale a quella spedita, allora si potrà proseguire con la creazione del file. In caso contrario il ricevente mostrerà un messaggio di errore, ma non forzerà il sender a rispedire il file. In questo modo si evita che il server debba gestire eventuali errori di trasmissione, oltre che ad evitare un eventuale loop causato da un continuo invio di pacchetti corrotti e richieste di riscaricare.

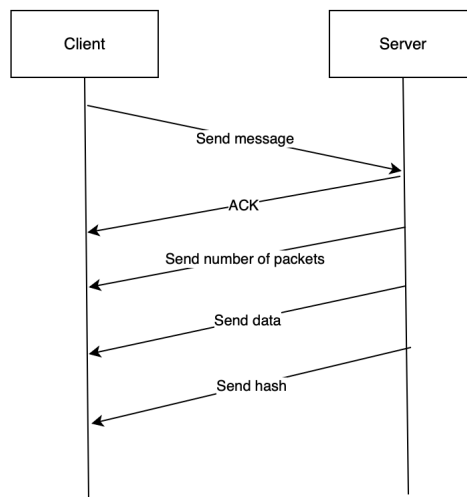


Figura 2.1: Get message.

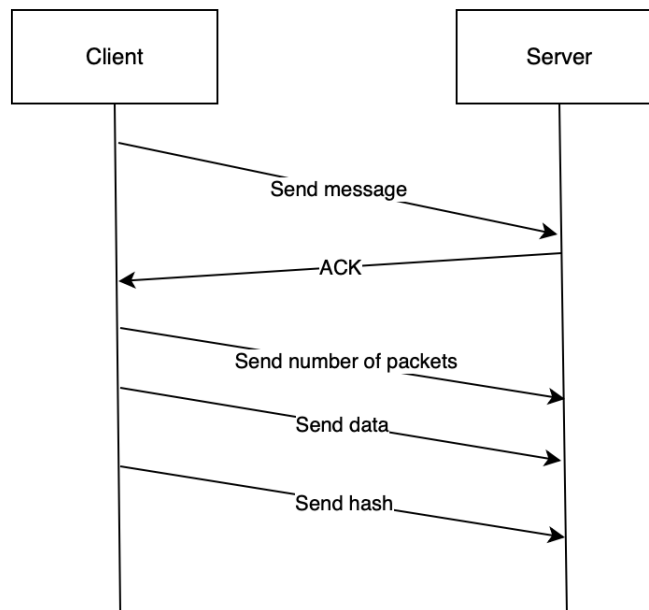


Figura 2.2: Put message.

2.2 Client e Server

Come menzionato inizialmente, il client e il server sono stati sviluppati nei relativi file .py, e mi sono servito di "utils.py" per raggruppare variabili globali comuni ad entrambi i file.

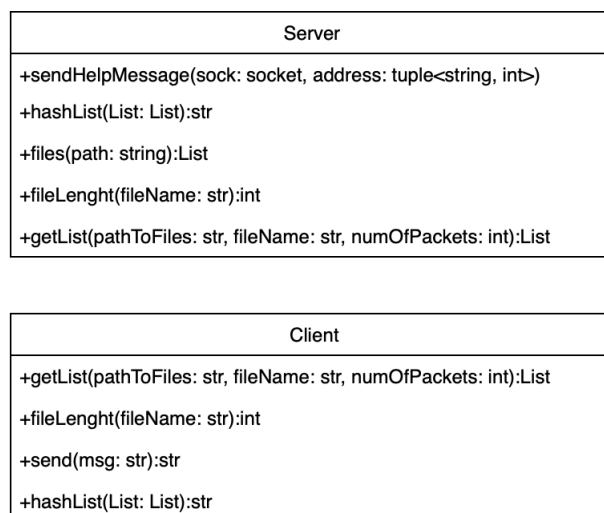


Figura 2.3: Organizzazione server e client.

Ogni metodo è ben spiegato dalla relativa documentazione. Ho deciso di non usare ulteriori classi o file in quanto è stato più facile e chiaro per me utilizzare questa struttura.

2.2.1 Limitazioni

Il server gestisce una richiesta alla volta; questo permette che più client siano connessi contemporaneamente ma le loro richieste non si devono sovrapporre.

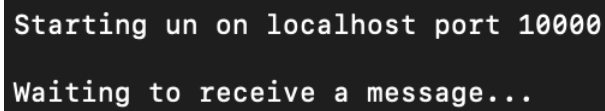
2.3 Timeout

Per non fare bloccare in una attesa infinita una delle due parti, ho implementato un semplice timeout di 5 secondi. Questo è presente durante la connessione tra client e server, ma anche quando vengono scambiate le richieste. Nello specifico, il server presenta un timeout solo quando il client sta facendo un operazione di "put", e quest'ultimo smette di inviare i pacchetti (esempio disconnessione del client). Il client invece presenta questa caratteristica ogni volta che fa una richiesta al server (comandi "list", "put", "get", "help"), e questo non risponde (esempio disconnessione del server).

Capitolo 3

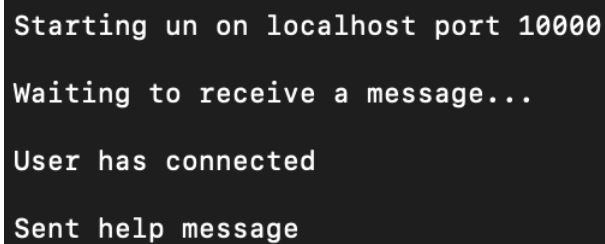
Guida utente

Da terminale, spostarsi dentro la cartella del progetto e in una console digitare "python3 server.py", poi in un'altra console digitare "python3 client.py". Fatto ciò basterà seguire le indicazioni a video che il client propone.

A terminal window with a dark background and light green text. The text displayed is "Starting un on localhost port 10000" followed by "Waiting to receive a message...". A small vertical bar is visible at the end of the second line, indicating the cursor position.

```
Starting un on localhost port 10000
Waiting to receive a message...
```

Figura 3.1: Server avviato, ma nessun client connesso.

A terminal window with a dark background and light green text. The text displayed is "Starting un on localhost port 10000", "Waiting to receive a message...", "User has connected", and "Sent help message". A small vertical bar is visible at the end of the fourth line, indicating the cursor position.

```
Starting un on localhost port 10000
Waiting to receive a message...
User has connected
Sent help message
```

Figura 3.2: Server avviato con un client connesso.

```
Connecting to server...
Hello, please type a command from the followings:

List -> view the files in the archive
Get "filemane" -> downloand the file from the archive
Put "filemane" -> upload the file to the archive
Quit -> exit the program
Help -> show this message again

Write a command:
█
```

Figura 3.3: Client avviato.