Progetto di Reti

Samuele Carafassi
Mat. 0000975004
samuele.carafassi@studio.unibo.it

24 luglio 2022

Sommario

Il progetto riguarda la costruzione di un server ed un client che attraverso il protocollo di livello transport UDP ed un opportuno protocollo di comunicazione il client può: richiedere la lista di file presenti su server, scaricare un file dal server o caricare un file sul server.

Indice

1 Descrizione progetto **2**
1.1 Scopo del progetto 2
1.2 Requisiti software 2
1.3 Funzionalità server 3
1.4 Funzionalità client 3

2 Progettazione **4**
2.1 Operazione GET 4
2.2 Operazione LIST 5
2.3 Operazione PUT 6
2.4 Comando HELP 6
2.5 Comando QUIT 6

3 Architettura generale **7**
3.1 Struttura directory progetto 7
3.2 Server 7
3.3 Client 7

4 Manuale d'uso **8**
4.1 GET 8
4.2 HELP 8
4.3 LIST 8
4.4 PUT 9
4.5 QUIT 9

Capitolo 1

Descrizione progetto

1.1 Scopo del progetto

Lo scopo de progetto è quello di progettare ed implementare in linguaggio Python un'applicazione client-server per il trasferimento di file che impieghi il servizio di rete senza connessione (socket tipo SOCK_DGRAM, ovvero UDP come protocollo di strato di trasporto).

1.2 Requisiti software

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;

La comunicazione tra client e server deve avvenire tramite un opportuno protocollo. Il protocollo di comunicazione deve prevedere lo scambio di due tipi di messaggi:

- messaggi di comando: vengono inviati dal client al server per richiedere l'esecuzione delle diverse operazioni;
- messaggi di risposta: vengono inviati dal server al client in risposta ad un comando con l'esito dell'operazione.

1.3 Funzionalità server

Il server deve fornire le seguenti funzionalità:

- L'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.

1.4 Funzionalità client

I client deve fornire le seguenti funzionalità:

- 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.

Capitolo 2

Progettazione

2.1 Operazione GET

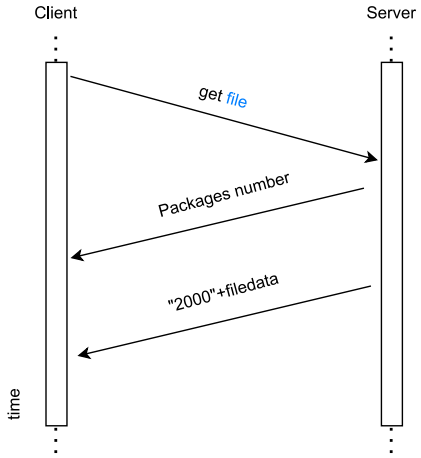

Figura 2.1: Schema richiesta get

Il client invia una richiesta GET di un determinato file al server. Il server controlla tra le sue risorse se il file è presente e risponde col numero di pacchetti che stanno per essere inviati al client. Se il file è presente verrà inviato il codice "2000" seguito dal file richiesto. Se il file non è presente il server risponderà con codice "1001" e relativo messaggio di errore.

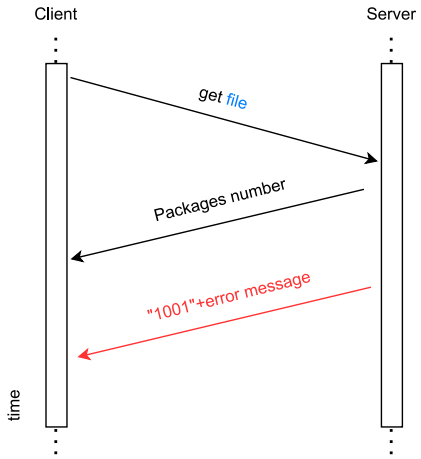

Figura 2.2: Schema richiesta get con errore

2.2 Operazione LIST

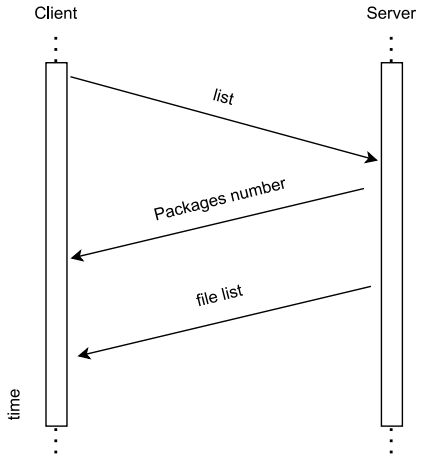

Figura 2.3: Schema richiesta list

Il client invia una richiesta LIST al server che risponderà col numero di pacchetti in arrivo (implementato per il caso in cui caso in cui il server abbia molti file tra le sue risorse) e la lista delle risorse a sua disposizione.

2.3 Operazione PUT

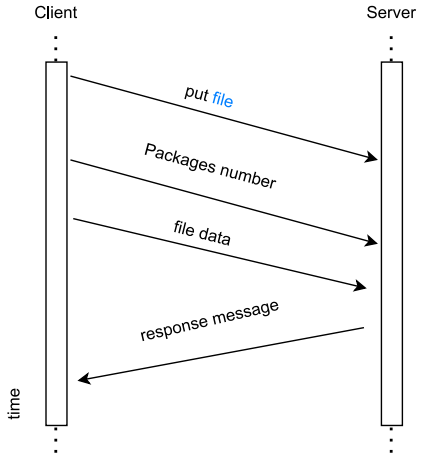

Figura 2.4: Schema richiesta put

Il client controlla tra le sue risorse se il file esiste. Se il file non esiste il client mostra un messaggio di errore, altrimenti invia una richiesta di PUT al server e il numero di pacchetti che gli saranno inviati. Il server riceve tutti i pacchetti e crea il file con i dati forniti dal client. In caso di esito positivo della creazione del file il server invierà un messaggio di successo, altrimenti invierà un messaggio di errore.

2.4 Comando HELP

Dal client si può digitare questo comando per mostrare i comandi e le operazione possibili.

2.5 Comando QUIT

Dal client si può digitare questo comando per chiudere il socket di connessione e terminare il client.

Capitolo 3

Architettura generale

3.1 Struttura directory progetto

Sono stati separati i file di codice dai file generici usati per testare get e put.
Nella cartella *src* si trovano i file "client.py" e "server.py" contenenti le classi "UDP_Client" e UDP_Server".
La cartella *res* è stata divisa in due sottocartelle: client e server. La cartella server è la cartella che contiene i file "presenti" sul server. La cartella client è la cartella che contiene i file scaricati dal server e quelli che può caricare sul server.

3.2 Server

Il server è rappresentato dalla classe UDP_Server che una volta istanziata crea un socket all'indirizzo 127.0.0.1 (localhost) sulla porta 10000.
Il server invia pacchetti di circa 32 Kb ed ha un buffer di ricezione della stessa dimensione. Tra l'invio di un pacchetto e il successivo ha un time-out di 0.1 secondi, da cui si può ricavare la velocità che è circa 320 Kb/s.
Il server mostra su console alcuni messaggi che vengono usati per controllare lo stato del server.

3.3 Client

Il client è rappresentato dalla classe UDP_Client che viene istanziata passandogli l'indirizzo (localhost) e la porta (10000) del server a cui deve connettersi.
Il client invia pacchetti di circa 32 Kb ed ha un buffer di ricezione della stessa dimensione. Tra l'invio di un pacchetto e il successivo ha un time-out di 0.1 secondi, da cui si può ricavare la velocità che è circa 320 Kb/s.

Capitolo 4

Manuale d'uso

Avviare "server.py" e "client.py". Nelle sezioni successive vengono illustrate le sintassi dei comandi del client.

4.1 GET

get *filename*

Comando usato per scaricare un file da server. *filename* rappresenta il nome del file e deve coincidere esattamente col nome del file effettivo (spazi, maiuscole, minuscole, segni di punteggiatura, ...). È possibile scaricare un solo file per volta.

4.2 HELP

help

Comando usato per ricevere la lista di comandi eseguibili.

4.3 LIST

list

Comando usato per ricevere la lista di file disponibili sul server.

4.4 PUT

put *filename*

Comando usato per caricare un file sul server. *filename* rappresenta il nome del file e deve coincidere esattamente col nome del file effettivo (spazi, maiuscole, minuscole, segni di punteggiatura, ...). È possibile caricare un solo file per volta.

4.5 QUIT

quit

Comando usato per chiudere il socket e terminare il client.