RELAZIONE PER PROGETTO DI PROGRAMMAZONE DI RETI

Traccia 2 – Trasferimento di file tramite UDP

Gabriele Menghi

gabriele.menghi@studio.unibo.it

0000997541

Indice

1. Esecuzione

Per testare il file transfer può essere utilizzata la riga di comando, avendo una versione di python installata nel computer, oppure è possibile utilizzare direttamente il compilatore messo a disposizione dall'IDE (e.g. Spyder). In entrambi i casi sarà necessario prima avviare il server (file *server.py* nella directory *Server*). Così facendo il server si metterà in ascolto sull'opportuna porta, stampando in output il relativo messaggio. A questo punto sarà possibile avviare anche il client (file *client.py* nella directory *Client*). Per eseguire un file da riga di comando è necessario spostarsi nell'apposita directory, digitare la versione di python seguita dal nome del file da eseguire (e.g. python3 server.py). Per eseguire un file da un IDE è necessario fare click destro sul file è scegliere l'opzione *run*.

1.1 Comandi

I comandi disponibili eseguibili dal client sono

- **list**: stampa l'elenco dei file disponibili nella directory del server (quindi anche lo stesso *server.py*);
- **get <filename>**: consente di scaricare il file indicato tra '<>' all'interno della directory del client (il nome del file deve corrispondere esattamente ad uno di quelli presenti nel server);
- **put**: apre una piccola finestra, cliccando la quale si apre il file explorer, che è possibile scorrere per scegliere un file (qualsiasi formato è accettato) da caricare sul server (all'interno della specifica directory);
- **exit**: chiude il socket del client che ha lanciato il comando.

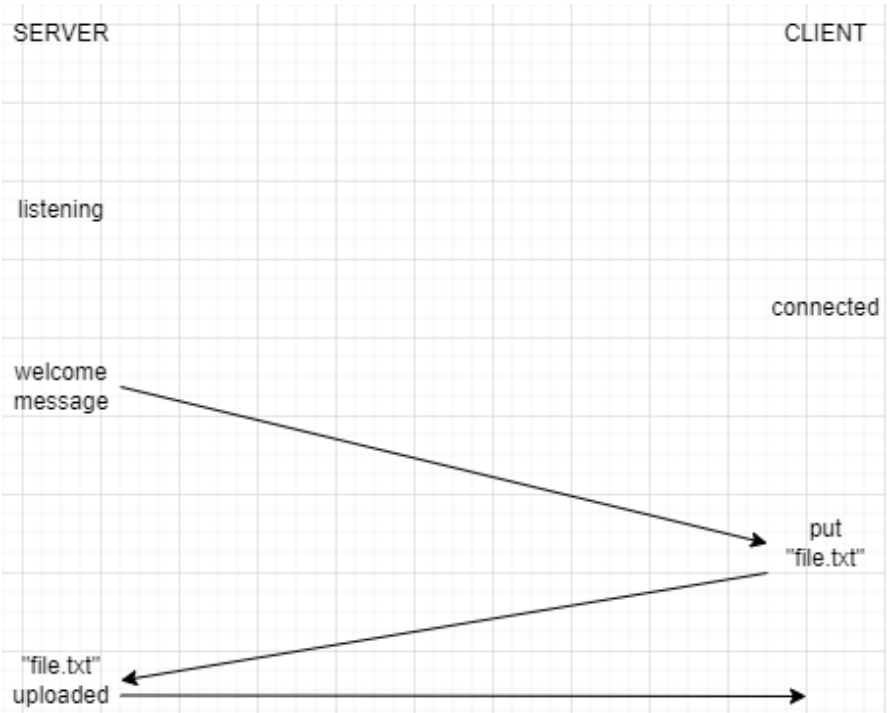

Immagine 1.1: schema rappresentante il comando put

2. Scelte di progetto

2.1 Struttura dei file e delle directory

Si è scelto di inserire i file *server.py* e *client.py* in apposite directory, in modo che i vari file caricati/scaricati vengano organizzati nelle medesime, senza rendere più complessa la struttura generale.

2.2 Variabili e funzioni

Vengono utilizzate tre variabili, la cui funzione principale è di evitare troppe ripetizioni. La prima (*size*) rappresenta la dimensione del buffer utilizzata, che si è scelta di 4096 bytes. È stata scelta questa soluzione dato il tipo di progetto, nel quale i protagonisti sono i file: con un buffer più grande è necessario spezzettare meno spesso i file, ed una maggiore quantità di dati vengono spediti simultaneamente.
La seconda variabile (*check*) serve a gestire l'invio di un file, per capire quando i dati contenuti nel file sono stati completamente trasferiti.
Infine la terza (*time_sleep*) è quella che impedisce che vengano persi pacchetti per strada, perché fa aspettare chi sta inviando il file una piccolissima frazione di secondo prima di inviare un pacchetto. In questo modo il ricevente ha tempo di ricevere tutti i pacchetti nel modo corretto, senza perdere nulla.
Sono state definite tre funzioni per il server e quattro per il client, con lo scopo di organizzare meglio il codice e renderlo più leggibile e comprensibile. Queste funzioni contengono il codice che gestisce i comandi disponibili ogni qualvolta vengano richiamati. C'è un'ultima funzione all'interno del client che permette l'apertura della finestra dalla quale è possibile attivare il file explorer a seguito della digitazione del comando put.

2.3 Messaggi e feedback

Vengono stampati a video messaggi contenenti esiti sulle operazioni nei seguenti casi

- richiesta da parte del client di un file non presente sul server
- inizio di scaricamento di un file dal server
- completamento dello scaricamento di un file dal server
- errore nell'apertura di un file da caricare sul server
- richiesta del comando put senza indicare nessun file (chiudendo il file explorer)
- inizio di caricamento di un file sul server
- corretto caricamento di un file sul server

Inoltre ogni volta che viene avviato il server, questo stampa un messaggio indicante che esso è attivo, specificando su quale porta. Infine per la maggior parte dei comandi, una volta che questo viene portato a termine, viene anche ristampata la lista di comandi disponibili (messaggio di benvenuto).

2.4 Socket

Per la gestione di alcuni comandi si è utilizzato un timeout sul socket, rimosso poi alla fine del comando stesso. Questo è stato utile a gestire i flussi di dati, per verificare quando fosse terminata la trasmissione dei pacchetti utili.
La non chiusura del socket del server è data dalla connessione alla base del file transfer, cioè una connessione basata su un protocollo di strato di trasporto di tipo UDP.
Può essere chiuso un socket del client, per lasciare spazio ad un client nuovo, ma anche in questo caso non si tratta di una vera e propria chiusura del socket, come avviene invece nel protocollo TCP.
Essendo la connessione di tipo UDP, sono state utilizzate di conseguenza le funzioni tipiche dei socket UDP.

2.5 File explorer

Per quanto riguarda il file aperto tramite file explorer del quale si vuole fare il put, questo nel server viene splittato attraverso il carattere '/' , in modo da ottenere il nome e l'estensione con le quali si vuole aggiungere il file nella directory del server. Questo carattere utilizzato nello split va bene per tutti i sistemi operativi perché è il carattere utilizzato dalla libreria tkinter, che gestisce nello stesso modo i file, astraendo dal sistema operativo. Questa restituirà sempre il percorso assoluto del file scelto nel file explorer usando '/' come separatore.

2.6 File

È possibile caricare/scaricare ogni tipo di file sul/dal server. Quando viene scelto un file da caricare tramite file explorer, viene salvato nel server con il nome attuale seguito dal formato. Se il nome risulta essere più lungo di cinquanta caratteri, vengono automaticamente selezionati gli ultimi cinquanta (compreso quindi il formato).

3. Strutture dati

L'unica struttura dati utilizzata nel progetto è la tupla, in questo caso formata da due elementi. È utilizzata per ospitare la coppia dati – indirizzo ottenuta dalla funzione *recvfrom()* di un socket.

4. Threads

Si è deciso di non utilizzare threads all'interno di questo progetto. Si è valutato se introdurli per gestire le singole funzioni rappresentanti i vari comandi, scartando poi questa opzione. Questo perché se si fossero mantenuti i threads, più comandi sarebbero potuti essere lanciati contemporaneamente, provocando possibili fraintendimenti nel ricevente, dato che i pacchetti sarebbero potuti essere di appartenenza di file o comandi diversi, generando un errore nel file ricostruito a partire dagli stessi pacchetti ricevuti, o un errore nel messaggio da stampare a video.