Relazione Progetto – Programmazione di Reti

Traccia 2

Andrea Dotti – matricola 0000970370
andrea.dotti4@studio.unibo.it

Giacomo Pierbattista – matricola 0900069485
giacomo.pierbattista@studio.unibo.it

Introduzione

Questo progetto è lo svolgimento della traccia 2, la quale consiste nella realizzazione di un'applicazione client - server che utilizza il protocollo UDP per trasferire file. Questa applicazione permette di

- Visualizzare la lista dei file disponibili sul server
- Scaricare file dal server al client (download)
- Inviare file dal client al server (upload).

Il repository del progetto si trova all'indirizzo

https://github.com/jackprb/UDP-client-server

Struttura del progetto

Il progetto è costituito dai seguenti file, organizzati come mostrato dall'immagine seguente:

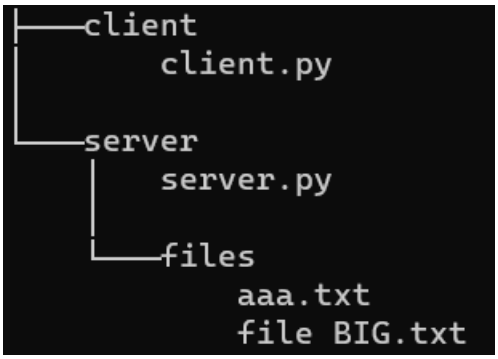

Figura 1: Struttura di file e cartelle dell'applicazione client – server.

Nella directory "server", la sottocartella "files" contiene i file che saranno resi disponibili dal server stesso, mentre nella directory "client", al primo avvio, viene creata una sottocartella "files" in cui verranno salvati i file scaricati dal server.

Avvio dell'applicazione

Per avviare l'applicazione, è necessario avviare separatamente, in due terminali distinti, il file server.py e, successivamente, il file client.py, come segue:

```
python server.py <numero porta>
```

```
python client.py 127.0.0.1 <numero porta>.
```

Dopo aver avviato il server e il client, saranno disponibili tutte le operazioni richieste.

Dettagli implementativi

Il client ha una CLI “user friendly”, che all’avvio mostra tutte le operazioni possibili, riconoscendo eventuali comandi mal formattati o sconosciuti.

```
Asking for connection...

Connection established

WELCOME!

Server is running at:   127.0.0.1:6000

List of all available commands:
help                 : show this help
clear                : clear the terminal
server               : print server IP and port
get <filename>       : downloads the specified file, if available on server
put <path_to_file>   : uploads the specified file to the server
list                 : shows list of all files available on server
shutdown             : switch off both server and client
end                  : close connection to the server

Type in a command:
>
```

Figura 2: CLI mostrata all’avvio del terminale che esegue il file client.py, riportante la descrizione delle operazioni permesse

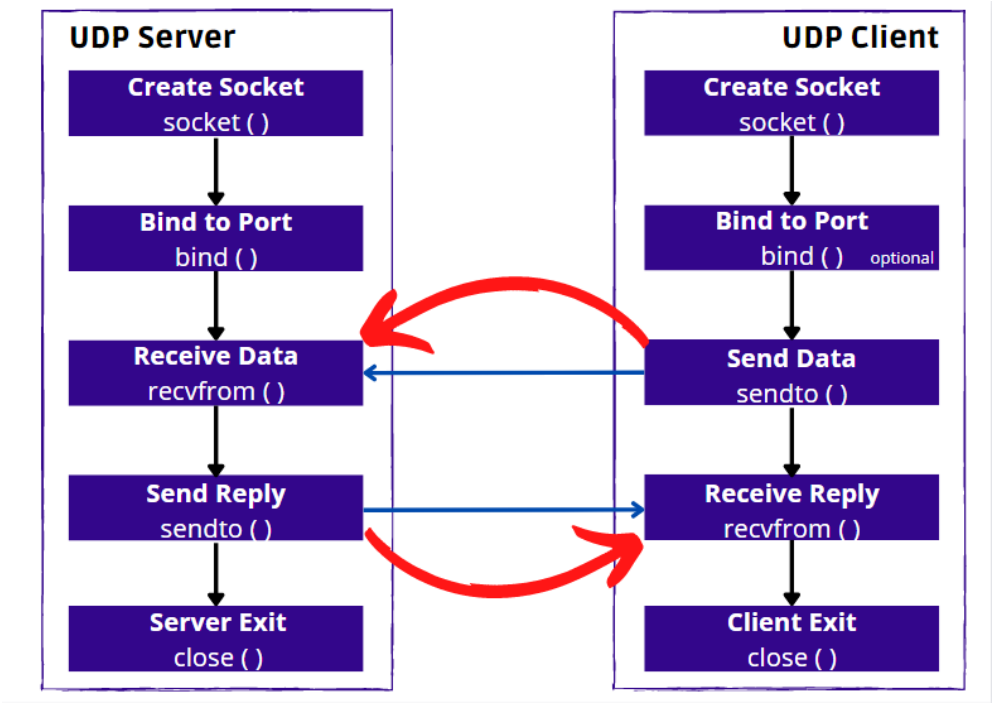

Figura 3: Schema dei socket

RTP

Nelle funzioni `list`, `get` e `put` si utilizza un protocollo RTP (Reliable Transfer Protocol), nel quale, per ogni pacchetto,

1. il mittente invia il pacchetto
2. il destinatario lo riceve, calcola l'MD5 di tale pacchetto, poi lo invia al mittente
3. il mittente controlla che l'MD5 ricevuto dal destinatario sia corretto:
   a. se l'MD5 corrisponde, il mittente invierà il messaggio "ok"
   b. altrimenti, il mittente invierà il messaggio "corrupted" e provvederà a inviare di nuovo tale pacchetto corrotto.

Funzione `list`

La funzione `list` ottiene la lista dei file disponibili sul server scaricabili dal client. Per ogni file disponibile sul server, si mostra il nome e la sua dimensione in byte.

Il server, attraverso la funzione `getAllFiles()`, crea una stringa formattata contenente nome e dimensione in byte di tutti i file disponibili sul server, come mostrato dall'immagine seguente.

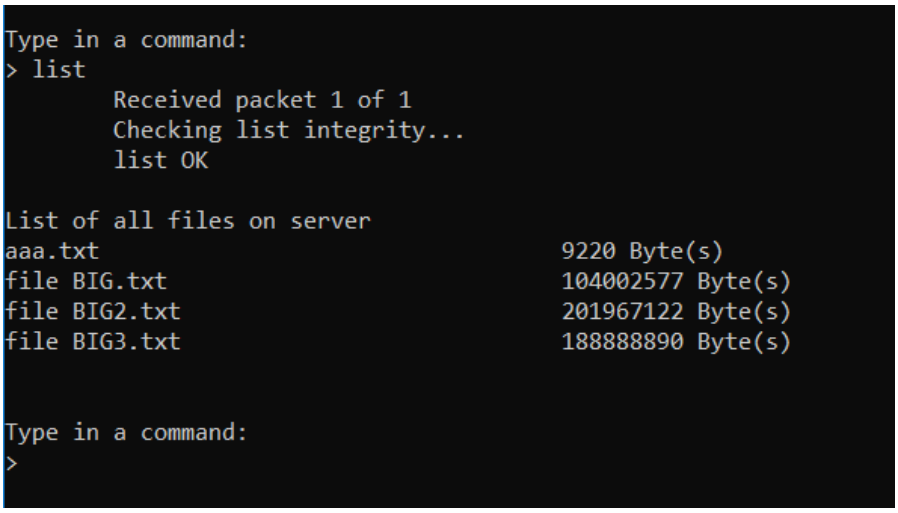

Figura 4: Lista dei file disponibili sul server con relativa dimensione in byte

Prima di inviare tale lista al client, si calcola il numero di pacchetti necessario per trasferirla, in quanto può capitare che la dimensione della stringa di cui sopra superi la dimensione massima stabilita per i pacchetti (32768 byte); in questo modo, come

illustrato dall'immagine seguente, si può inviare una lista di dimensioni arbitrarie (che richiederà l'invio di un numero arbitrario di pacchetti).

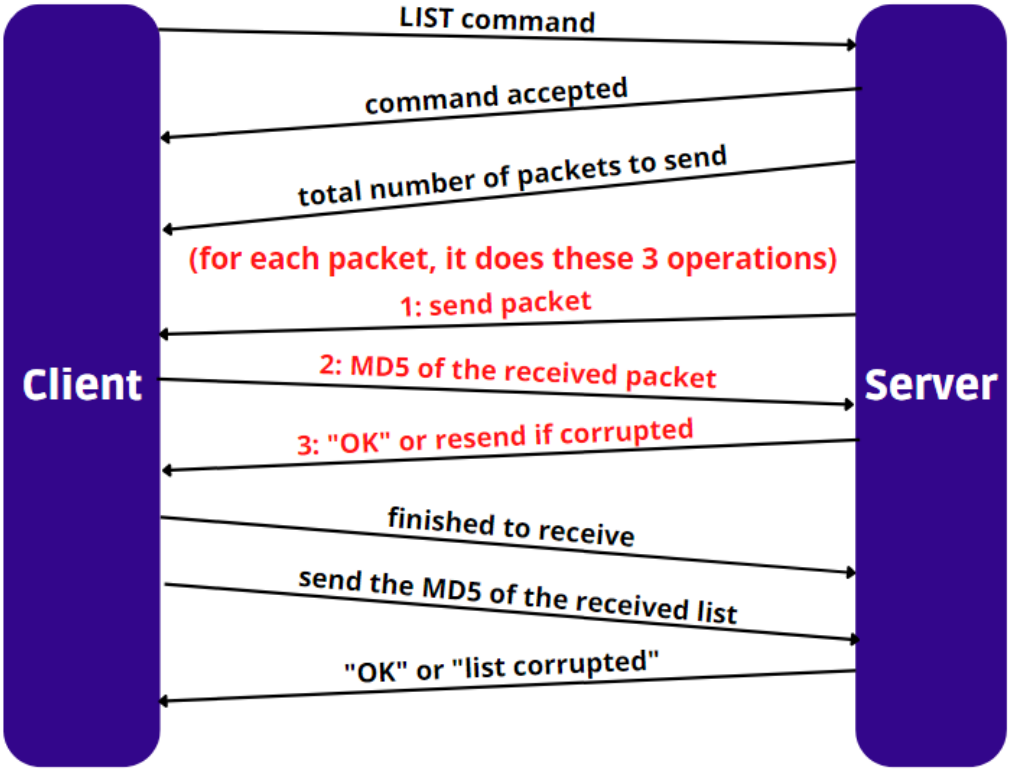

Figura 5: Schema di messaggi inviati tra client e server nella funzione LIST

Quando il client invia il comando list, se ben formattato e senza errori, il server risponderà con un messaggio "command accepted" e invierà il numero totale di pacchetti (N) necessari per inviare tutta la lista dei file.

Dopodiché, per ognuno degli N pacchetti, si applica il protocollo RTP descritto in precedenza per garantire che ognuno di essi arrivi a destinazione non danneggiato.

Quando il client ha ricevuto con successo tutti i pacchetti, invia un messaggio "finished" al server e successivamente invia l'MD5 della lista complessiva ricevuta. Se tale MD5 corrisponde con quello previsto, il server invia il messaggio "ok", "corrupted" altrimenti.

In quest'ultimo caso, il client mostrerà un messaggio di errore e l'utente dovrà inviare nuovamente il comando list per ottenere la lista dei file.

Funzione `get`

La funzione `get` permette al client di scaricare uno dei file disponibili sul server (download). Se il trasferimento si completa con successo, il file ricevuto viene salvato nella cartella "Client\files". Se si scarica un file con lo stesso nome di uno già presente nella directory "files", quest'ultimo verrà sovrascritto.

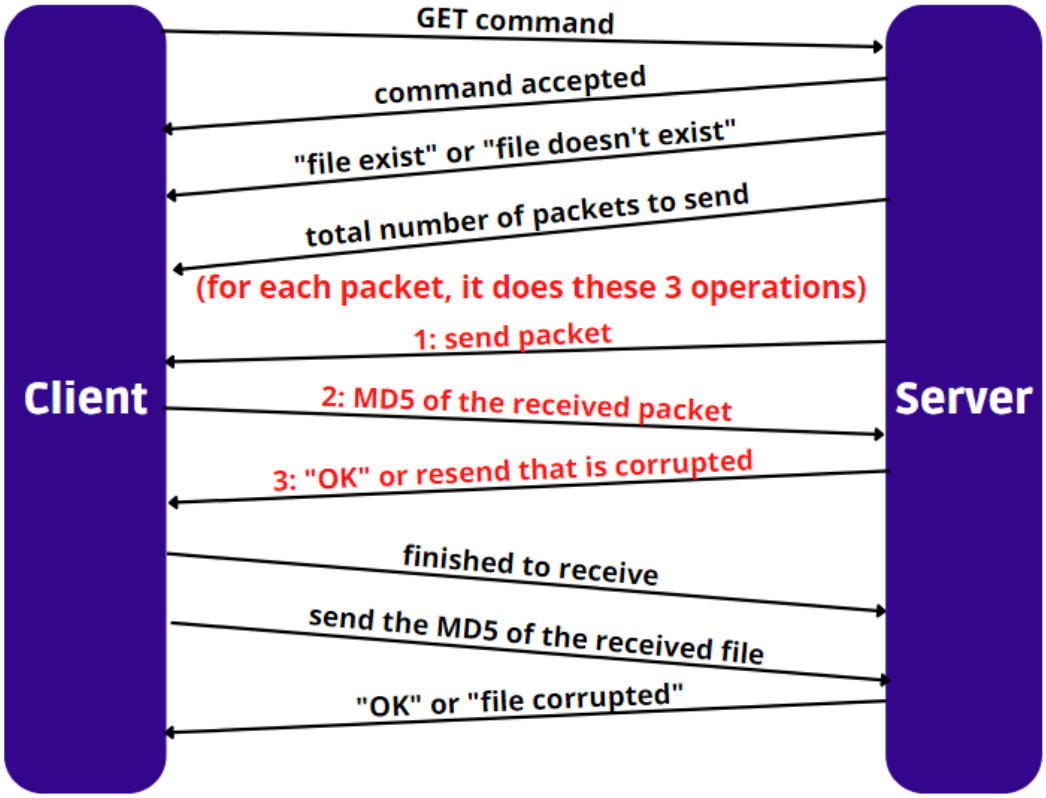

Figura 6: Schema di messaggi inviati tra client e server nella funzione GET

Quando il client invia il comando `get`, se ben formattato e senza errori, il server risponderà con un messaggio "command accepted" e invierà il numero totale di pacchetti (N) necessari per inviare tutto il file.

Dopodiché, per ognuno degli N pacchetti, si applica il protocollo RTP descritto in precedenza per garantire che ognuno di essi arrivi a destinazione non danneggiato.

Quando il client ha ricevuto con successo tutti i pacchetti, invia un messaggio "finished" al server e successivamente invia l'MD5 del file ricevuto. Se tale MD5 corrisponde con quello previsto, il server invia il messaggio "ok", "corrupted" altrimenti.

In quest'ultimo caso, il client mostrerà un messaggio di errore, cancellerà il file corrotto ricevuto e l'utente dovrà inviare nuovamente il comando `get <nomeFile>` per ottenere il file.

Funzione `put`

La funzione `put` permette al client di inviare un file al server (upload); se il trasferimento si completa con successo, il file inviato viene salvato nella cartella "Server\files", e da questo momento in poi comparirà nella lista dei file scaricabili. Se si invia un file con lo stesso nome di uno già presente nella cartella "Server\client", quest'ultimo verrà sovrascritto.

Quando si carica un file, se esso si trova nella stessa cartella del file "client.py" è sufficiente specificare solo il nome del file stesso, in tutti gli altri casi è necessario specificare il percorso assoluto del file.

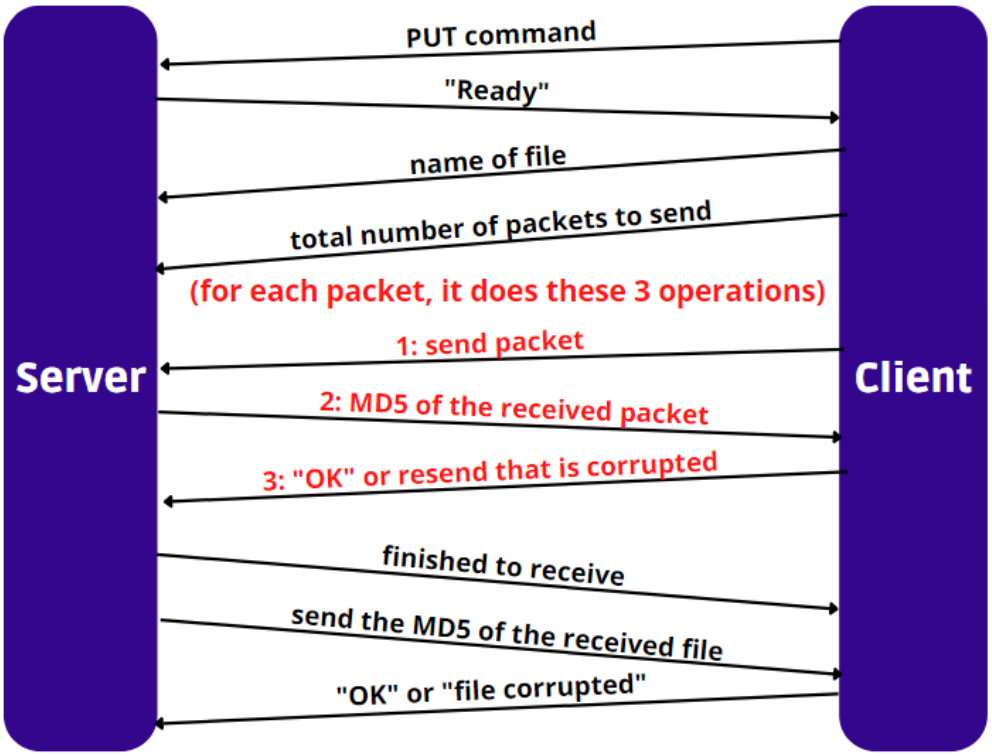

Figura 7: Schema di messaggi inviati tra client e server nella funzione PUT

Quando il client invia il comando `put`, se ben formattato e senza errori, il server risponderà con un messaggio “ready”; successivamente il client invierà il nome del file, poi il numero totale di pacchetti (N) necessari per inviare tutto il file.

Dopodiché, per ognuno degli N pacchetti, si applica il protocollo RTP descritto in precedenza per garantire che ognuno di essi arrivi a destinazione non danneggiato.

Quando il server ha ricevuto con successo tutti i pacchetti, invia un messaggio “finished” al client e successivamente invia l’MD5 del file ricevuto. Se tale MD5 corrisponde con quello previsto, il client invia il messaggio “ok”, “corrupted” altrimenti.

In quest’ultimo caso, il client mostrerà un messaggio di errore e cancellerà il file corrotto ricevuto e l’utente dovrà inviare nuovamente il comando `put <path_to_file>` per fare l’upload del file.

Comandi aggiuntivi lato client

Segue una breve descrizione dei comandi aggiuntivi disponibili lato client:

- **help**: mostra il messaggio di aiuto mostrato nella Figura 1
- **clear**: cancella tutto il contenuto testuale del terminale
- **server**: stampa su terminale l’indirizzo (IP e porta) su cui il server è in esecuzione
- **shutdown**: termina l’esecuzione del client e del server, chiudendo i rispettivi socket
- **end**: termina l’esecuzione del client chiudendo il relativo socket e interrompendo la connessione al server