Relazione di Programmazione di Reti

Riccardo Mingozzi

riccardo.mingozzi4@studio.unibo.it

Matricola 0000989170

1 Traccia progetto

La traccia scelta per questo progetto è la seconda, consisente nella creazione di un server UDP per la gestione di download ed upload di file. Le funzionalità richieste sono la gestione di trasferimenti di file in ambo i lati, la possibilità di visualizzare la lista di file disponibili da lato server e la gestione di eventuali errori.

2 Discussione progettuale

Essendo utilizzato il protocollo UDP è stata tenuta in considerazione l'alta probabilità di fallimento nei trasferimenti. Di conseguenza, oltre che per la pulizia del codice, sono state definite funzioni generali di upload e download (quasi identiche sia per client che server) che tengono conto passo per passo di possibili errori, gestendo l'eventuale inesistenza del file, il fallimento del trasferimento e la sua possibile corruzione. Per quest'ultima, il controllo viene effettuato tramite hashing. Viene adottata la SHA256 per la sua elevata sicurezza e facilità di implementazione in ambiente Python. Sia il lato client che il lato server sono inoltre arricchiti di vari messaggi informativi riguardati lo status di entrambi, in modo da permettere una comprensione chiara tramite terminale dei loro status. In particolare, essendo il client ad iniziare le comunicazioni, il server stampa sempre quale sia stato l'ultimo comando su terminale, rendendo sempre chiaro ed esplicito cosa abbia chiesto il client. La grandezza dei pacchetti trasferiti è stata scelta di 2048 byte, in quanto potenza non troppo grande del 2.

3 Comandi

Verranno ora analizzati i comandi esposti, corredati nella sezione successiva dei rispettivi diagrammi di flusso.

- Exit All'invio di questo comando, il client notifica il server, e successivamente chiude il socket e termina lo script. Alla ricezione della notifica del client, il server si comporta allo stesso modo, chiudendo il socket e terminando lo script.
- List Quando il client notifica il server con il comando list, questo crea una lista di tutti i file nella propria cartella, per poi in-

viarla. Una volta iniviata, il server la elimina, ed il client, una volta letta e stampata su terminale, la elimina a sua volta. È stato scelto questo approccio, poichè dovendo implementare download ed upload bidirezionale per file, questo comportava dovere solo creare un file (lato server) e farlo leggere (lato client).

- Put Inviato il comando put, il client notifica il server che si mette di conseguenza in attesa, e la comunicazione inizia, effettuando ogni step gli opportuni controlli. Inizialmente, se il file esiste. In caso negativo, viene fermato tutto prima ancora che inizi. Una volta individuato il file, viene inviato. Viene poi controllato alla file se è stato effettivamente scaricato e, in caso positivo, vengono confrontati i due hash, uno fornito dal client e l'altro calcolato sul momento dal server. Se combaciano, il file viene considerato effettivamente scaricato, altrimenti verrà eliminato.

- Get Quasi identico al comando Put, per via del riutilizzo dello stesso codice appositamente generalizzato il più possibile, l'unica differenza risiede solamente nel come viene iniziata la comunicazione, notificando il server e mettendosi in attesa di ricevere inizialmente conferma il file effettivamente esista e successivamente, una volta ricevuto il file, che passino tutti gli altri controlli.

4 Diagrammi di flusso

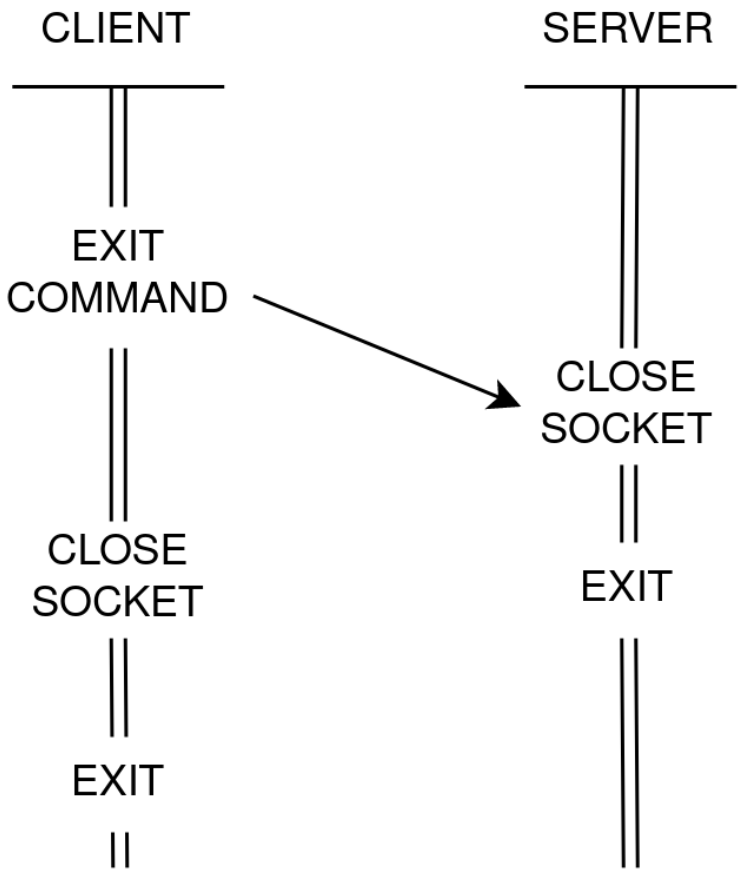

Figure 1: Flowchart of the exit command

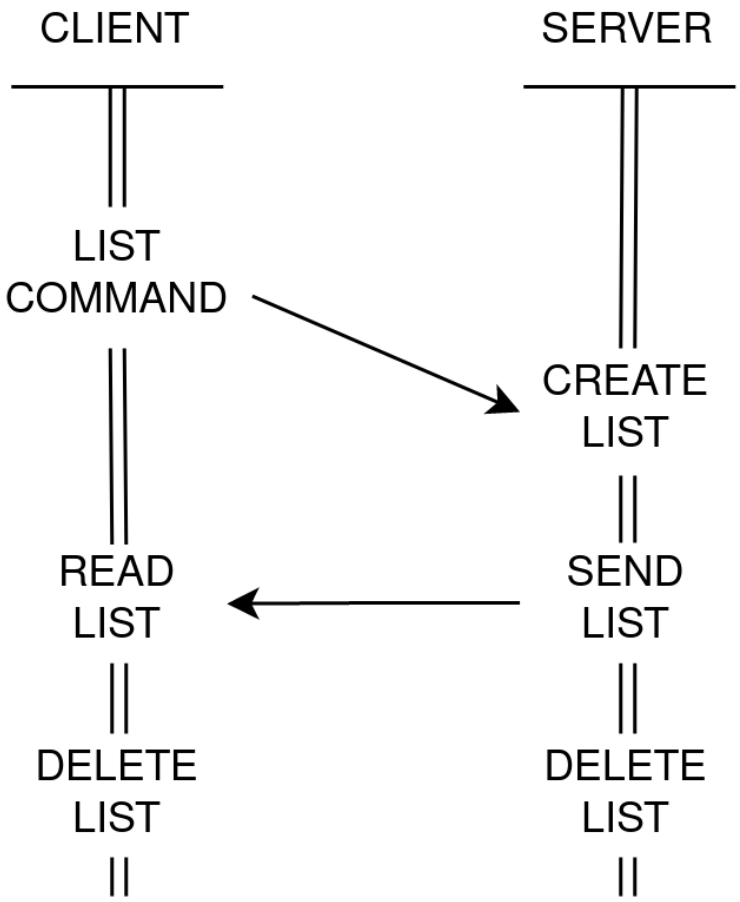

Figure 2: Flowchart of the list command

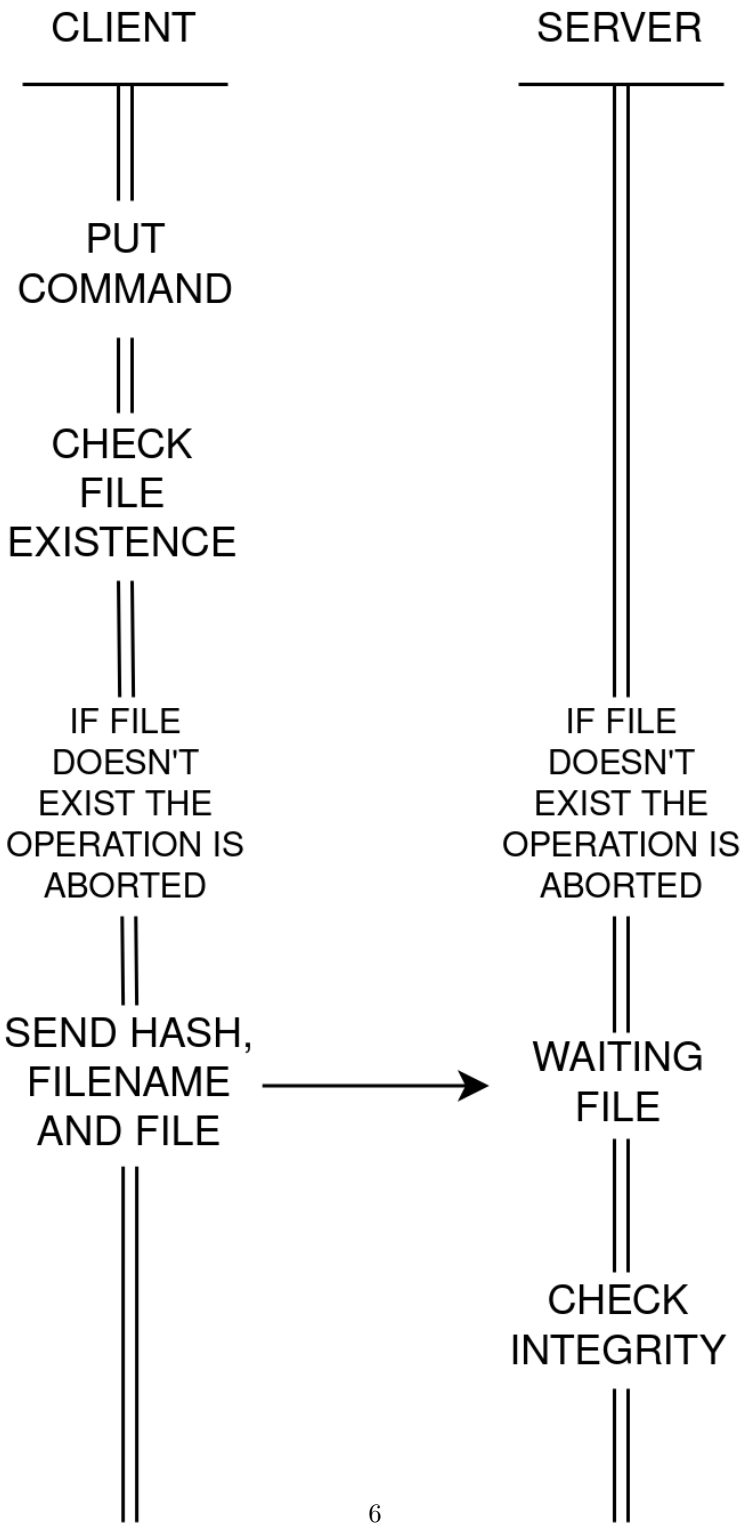

Figure 3: Flowchart of the put command

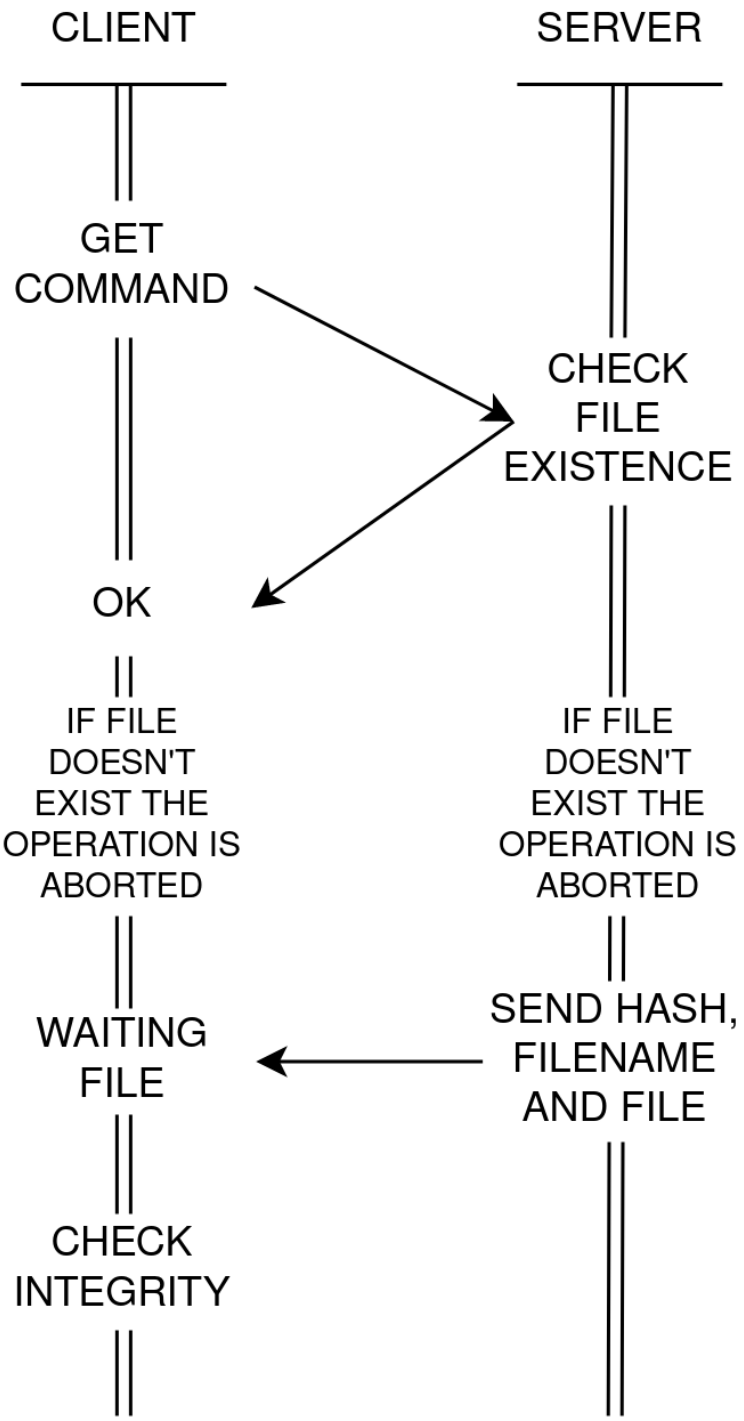

Figure 4: Flowchart of the get command