Relazione Progetto Programmazione di Reti 2022

Michele Montesi
Matricola: 0000974934
E-Mail: michele.montesi3@studio.unibo.it

21 giugno 2022

Indice

Capitolo 1

Analisi

Si é realizzata la traccia numero 2, la quale riguarda la creazione di un'architettura **Client-Server**, utilizzando il protocollo di strato di trasporto `UDP`, per il trasferimento di file. Deve essere possibile lo scambio di due tipi di messaggio:

- Messaggi di comando
- Messaggi di risposta

Le funzioni richieste sono:

- **LIST**: lista dei file contenuti all'interno del server.
- **GET**: comando di richiesta file dal server.
- **PUT**: upload di un file sul server.

Capitolo 2

Design

2.1 Panoramica

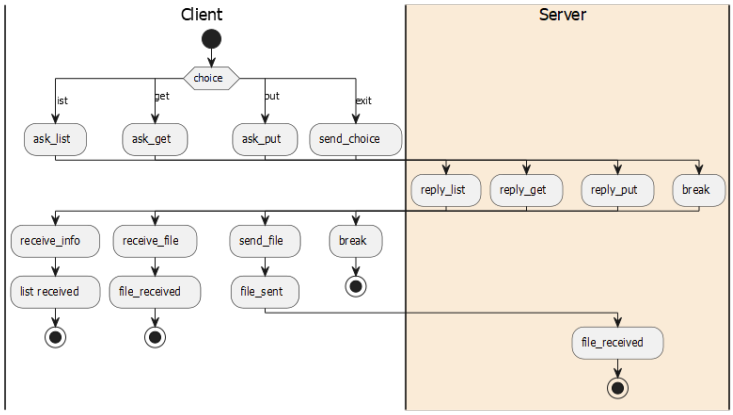

Figura 2.1: Rappresentazione minimale funzionamento Client Server

L'architettura é composta da due moduli: il modulo lato `client` ed il modulo lato `server`.

- Per quanto riguarda il lato client, é stata implementata una classe *client*, la quale fornisce ed espone in console le funzionalitá richieste.
- Il server, sta in ascolto, aspettando un input riguardante la funzione scelta dal client. Quando perviene la richiesta, il server avvia un thread circa l'operazione in questione.

Le funzioni in comune tra i due moduli sono stati inseriti in una classe di `utility` evitando cosí la ripetizione del codice.

2.2 Design dettagliato

2.2.1 Invio di un file da Client a Server

Per selezionare un file verrá fatta una verifica della sua esistenza lato client. Nel caso esista si proseguirá con l'invio, altrimenti verrá restituito un messaggio d'errore. Appurata l'esistenza di questo, verrá inviato al server il nome del file assieme alla sua dimensione, inviando una stringa. Questa é separata da una costante *Separator* la quale sará usata come divisore tra le due informazioni. Una volta arrivata al server, questo inizierá la scrittura di un file a cui verrá dato il nome del file originale con una *'r_'* davanti, questo permetterá di evitare l'*overwrite* di file originali del server ma la permetterá per file caricati in un primo momento, i quali dovranno poter essere aggiornati alla nuova versione.
Per compiere l'operazione di *put* é stato deciso di dividere il file in questione in blocchi da 1024 bytes.
Per seguire l'andamento dell'invio del file é stata inserita una barra di progressione divisa anch'essa in blocchi da 1024 bytes. Per completare l'invio del file verrá inviata una stringa contenente `file_shared_exit`. Il server quando leggerá questa stringa tra i bytes ricevuti, interromperá la ricezione e chiuderá il socket.

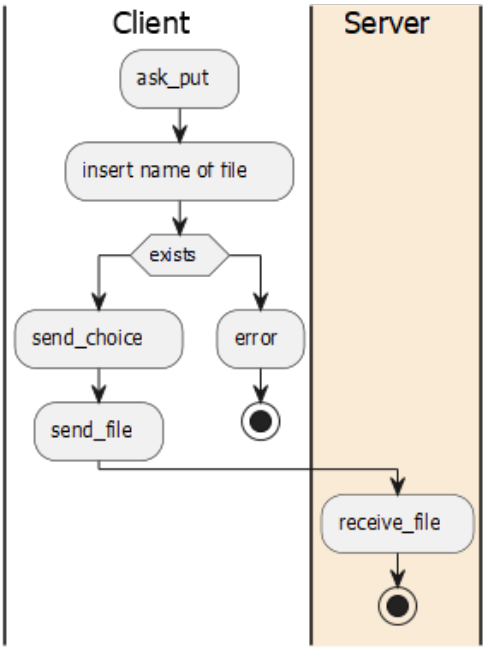

Figura 2.2: Invio di un file da Client a Server

2.2.2 Ricezione di un file da Server a Client

Il meccanismo di *get* é praticamente uguale al meccanismo di *put* con l'unica differenza che il controllo di esistenza del file viene eseguito sul server, il quale nel caso di esistenza consente l'invio del file, mentre in caso contrario restituirá un messaggio d'errore.
Nel dettaglio, al momento della richiesta il client informerá il server sull'operazione da eseguire, dopodiché il client fará richiesta all'operatore esterno di immettere il nome di un file esistente sul server. Una volta inserito, e confermato, questo verrá inoltrato al server, il quale ne controllerá l'esistenza. In caso affermativo, il server procederá all'invio come spiegato nel capitolo precedente (2.2.1)

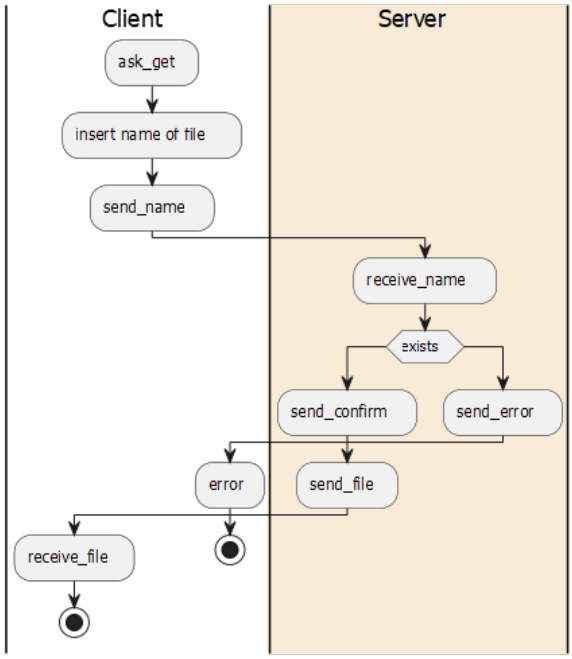

Figura 2.3: Invio di un file da Server a Client

Capitolo 3

Threads attivi

Per ogni funzione, sia lato Server che lato Client, viene avviato un Thread, in modo che il server possa comunicare con piú client.

3.1 Operazioni simultanee

Possibili

- *list* + *put* or *get*
- *put* + *get*

Non Possibili

- *put* + *put*
- *get* + *get*