Matricola: 0000970588
E-mail: anna.bartolucci@studio.unibo.it

Matricola:0000970591
E-mail: eleonora.falconi@studio.unibo.it

Relazione Progetto di Programmazione di Reti

Anno accademico 2021/2022

Architettura client-server UDP

Bartolucci Anna

Matricola: 0000970588

E-mail: anna.bartolucci@studio.unibo.it

Falconi Eleonora

Matricola: 0000970591

E-mail: eleonora.falconi@studio.unibo.it

Matricola: 0000970588
E-mail: anna.bartolucci@studio.unibo.it
Matricola:0000970591
E-mail: eleonora.falconi@studio.unibo.it

TRACCIA 2:

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).

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.

Funzionalità del ***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.

Funzionalità del ***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.

Dettagli per l'uso

Il progetto è composto da due applicazioni: Client.py e Serve.py. Per un corretto funzionamento del programma è necessario l'utilizzo dell'ultima versione di Python. Nell'eventualità si riscontrassero problemi con la compilazione lanciare i seguenti comandi dalla Powershell Prompt di Anaconda:

```
conda create -n spyder-cf -c conda-forge spyder
conda activate spyder-cf
spyder
```

Specifichiamo che dalla seconda volta in poi basta utilizzare solo gli ultimi due comandi.

Lanciare entrambi gli script su console, è possibile effettuare l'esecuzione di più client e un unico server in contemporanea.

Nel nostro programma l'interfaccia client può dispensare i seguenti comandi per gestire il lato server:

a) LIST: restituisce in output la lista dei nomi dei file presenti nella cartella "server_folder"
b) GET <fileName>: permette di salvare il file indicato nella cartella "client_folder"
c) PUT <path>: effettua l'upload di un file sul server specificandone il percorso viene salvato nella cartella "server_folder"
d) HELP: stampa a video tutti i possibili ordini che può impartire il client al server
e) EXIT: permette l'uscita dal programma

Le due cartelle relative alle interfacce sono rispettivamente: "client_folder" e "server_folder". Nel caso in cui le due cartelle non siano già presenti vengono create di default. Sia il client che il server saranno sempre attivi garantendo il servizio fin tanto che non si effettua l'uscita forzata dal programma. BUFFERSIZE rappresenta la dimensione costante dei pacchetti scambiati.

client.py

Allo scopo di creare il "server_address" nello script inizialmente viene generato il socket e vengono impostati l'indirizzo IP a "localhost" e la porta a "10.000".

Di seguito elenchiamo le varie funzioni gestite dal client:

- **LIST**: si occupa di richiedere al server la lista di file presenti nella cartella "server_folder". Nel caso in cui la cartella contiene almeno uno o più file si effettua la stampa a video dei rispettivi nomi, in caso contrario restituisce il messaggio di errore "Error: empty list".
- **GET <fileName>:** viene utilizzata per inviare la richiesta di Download del file specificato in "client_folder". Innanzitutto si controlla l'esistenza del file all'interno della cartella "client_folder", in caso sia già presente viene segnalato l'errore "Error: file already downloaded on the directory". Altrimenti se il file specificato è presente nella cartella "server_folder" si procede con l'apertura in scrittura e se l'operazione va' a buon fine si visualizza sulla console il messaggio "Finished Download", diversamente viene lanciato l'errore "File not valid ...".
- **PUT <path>:** permette di caricare il file nella cartella "server_folder". Se il file non esiste nel "client_folder" si lancia l'errore "Error: not existing file". Contrariamente si aggiorna il server della scelta del client, si apre il file in lettura effettuando l'upload.
- **HELP**: stampa a video il messaggio del menù che contiene i possibili comandi da impartire.
- **EXIT**: serve per terminare l'esecuzione del programma che avviene chiudendo il socket e si stampa il messaggio di uscita.

server.py

Allo scopo di creare il "server_address" nello script inizialmente viene generato il socket e vengono impostati l'indirizzo IP a "localhost" e la porta a "10.000". Attraverso la funzione bind() viene associato il "server_address" al socket chiamato "sock_server", dopodiché quest'ultimo si mette in ascolto. È stato deciso di introdurre i thread con lo scopo di permettere a più client di effettuare richieste ad un unico server. Per ogni client viene stanziato un nuovo thread, inizializzato con la funzione start(). Attraverso la funzione join() si ha l'effetto di bloccare il thread corrente finché il target thread di cui è appena stata fatta la join non è terminato. Abbiamo realizzato la funzione "client_handler" per gestire le funzionalità del server in relazione a ciò che viene richiesto dal client.

Di seguito elenchiamo le varie funzioni gestite dal server:

- **LIST**: come specificato in precedenza se la cartella "server_folder" non è vuota il server invia i nomi dei file contenuti al client
- **SEND**: utilizzata in risposta al messaggio di GET impartito dal client. Se è presente nella directory del server effettua l'invio del messaggio "OK" al client e procede con l'apertura in lettura del file. Qualora non sia presente si spedisce il messaggio "no" e si visualizza l'errore "File doesn't exist".

- **RECEIVE**: quando il client seleziona PUT <path> il server controlla l'esistenza del file in "server_folder" dando l'errore "Error: file already exists" se il controllo risulta vero. Se invece il file non risulta nella cartella si effettua l'upload e lo si apre in scrittura.
- **EXIT**: arresta il software con la close() per il socket, si visualizza poi il messaggio di uscita

Schema

Architettura Client – Server UDP