RELAZIONE DI PROGETTO PROGRAMMAZIONE DI RETI

Traccia 2

Alesja Delja – Alesja.Delja@studio.unibo.it

0000969763

Silvia Furegato – Silvia.Furegato@studio.unibo.it

0000977475

Tommaso Turci – tommaso.turci5@studio.unibo.it

0000971189

Indice

Descrizione generale delle scelte di progetto effettuate

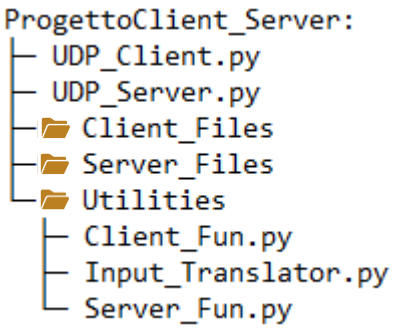
Per poter avviare il programma bisogna avere installato Python e aver clonato il repository https://github.com/AivAiv/ProgettoClient-Server. Con il terminale spostarsi fino alla cartella in cui si è clonato il repository e aprire una scheda di terminale.

Per eseguire il codice di server/client basta scrivere da terminale, nella cartella in cui sono presenti: python3 *nomefile*.py

Tutti i file che vengono condivisi tra il client e il server vengono salvati nelle cartelle Client_Files e Server_Files , che vengono autogenerate se non presenti.

Quando si desidera terminare il programma, è sufficiente digitare il comando "exit" sul client, che porterà all'arresto sia il client che il server.

La struttura directory è la sottostante:



Le cartelle Client_Files e Server_Files contengono i file coinvolti nel trasferimento. Abbiamo deciso di evitare l'implementazione di un sistema di riferimento ai file più flessibile (directory assolute/relative) in quanto ciò non ricadeva negli obiettivi evidenziati dalla traccia.

Funzionalità del client

Nel file UDP_Client vengono importato il socket, Input_Translator e Client_Fun che servono per implementare il Client. (Input_Translator e Client_Fun si trovano dentro la cartella Utilities.)

UDP_Client.py

Nel file creiamo un nuovo socket. Inoltre salviamo nella tupla server_address l' indirizzo IP 'localhost' e la porta 10000, la quale rappresenta un ipotetico server esterno. All'avvio del Client viene subito mostrato il server_address e la lista dei comandi che possono essere eseguiti; questo input viene poi controllato dall'Input_Translator importato precedentemente.

In base al comando scelto verrà poi eseguita l'operazione desiderata che si trova nel file Client_Fun. Infine, viene implementato anche il comando exit per terminare la connessione con il server e chiudere entrambe le esecuzioni.

Input_Translator.py

Questo file viene usato per controllare che gli input siano validi sfruttando diverse funzioni:

- getFiles (PATH)
  prende tutti i nomi dei file presenti nella directory PATH e li metti in una lista.

- checkInput(inString)
  controlla che il comando sia giusto indipendentemente che sia scritto in maiuscolo o minuscolo e controlla il nome del file. Inoltre, si assicura di controllare la directory corretta a seconda che l'operazione nella quale è convolta sia stata eseguita dal Client o dal Server.

- getInput(inString)
  ritorna il nome del file e il comando richiesto, dopo averli controllati tramite l'esecuzione di una checkInput(inString)

- split(inString)
  una semplice funzione di utility, suddivide il comando passato dal nome del file scelto.

Client_Fun.py

Implementa le diverse funzioni necessarie per il corretto funzionamento del client:

- ListRequest (socket, server_address)
  invia la richiesta List al server, e attende una risposta. Una volta ricevuta la lista dei file, la stampa a schermo.

- PutRequest (socket, server_address, filename)
  invia la richiesta Put al server insieme al file indicato, suddiviso in pacchetti, e attende una risposta. Una volta ricevuta la conferma che il trasferimento è avvenuto con successo o meno, stampa a schermo il risultato dell'operazione.

- GetRequest (socket, server_address, filename)
  invia la richiesta Get al server insieme al nome file che si desidera scaricare, e attende una risposta. Il server invia il file richiesto suddiviso in pacchetti, se presente. Una volta ricevuta la conferma che il trasferimento è avvenuto con successo o meno, stampa a schermo il risultato dell'operazione.

Funzionalità del server

Il server gestisce i propri file similarmente al Client; a differenza dell'ultimo, l'utente non è in grado di interagirci tramite input diretto.

UDP_Server.py

Nel file creiamo un nuovo socket e tramite l'istruzione bind() associamo il socket del server all'indirizzo IP 'localhost' e alla porta 10000. All'avvio del Server viene mostrato il server_address e confermata la disponibilità del Server a ricevere istruzioni tramite un messaggio in console.

A seconda della richiesta inviata dal Client verranno eseguite le rispettive funzioni di Server_Fun.py.

Server_Fun.py

Implementa le diverse funzioni che ci servono per il corretto funzionamento del server:

- ListResponse (socket, client_address)
  restituisce al Client la lista dei file correntemente presenti sul Server tramite l'utilizzo della funzione di Utility getServerFiles().

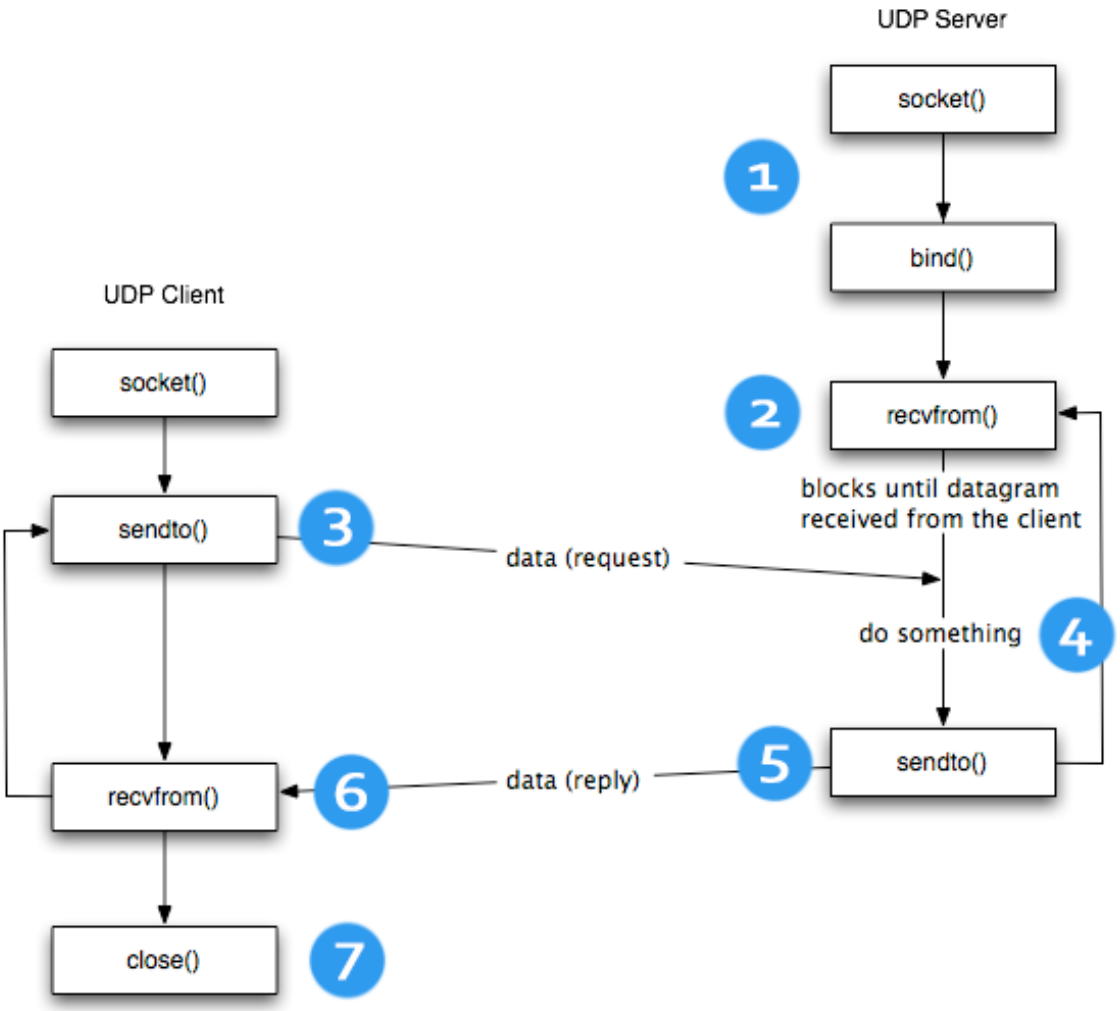
- PutResponse (socket, client_address)
  riceve i pacchetti inviati dal Client, ricomponendoli nel file originale. Per ciascuno esegue dei check di consistenza per evitare la perdita di pacchetti. In caso un pacchetto venga perso, viene richiesto al Client un nuovo invio dell'ultimo. Successivamente, il file viene salvato su disco nella directory Server_Files; infine, il Server conferma al Client il trasferimento.

- GetResponse (socket, server_address)
  similmente al funzionamento della funzione PutRequest() del Client, la GetResponse() invia al Client il file richiesto, suddiviso in pacchetti. La conferma di successo viene gestita dal Client.

- getServerFiles ()
  controlla se la cartella di destinazione esiste già, in caso contrario, la crea; successivamente raccoglie i nomi dei file in una lista, ignorando eventuali cartelle annidate per evitare errori.

Descrizione delle strutture dati utilizzate

Lo schema sottostante semplifica il funzionamento delle due esecuzioni:



Analizziamo i corrispondenti passaggi tramite la console:

1) Il socket del Server viene creato e associato all'indirizzo preimpostato;

```
sock = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
sock.bind(SERVER_ADDRESS)
```

2) Il server si mette in ascolto;

```
Server started and listening on IP: localhost and PORT: 10000
```

3) Il client invia una richiesta al server;

```
Insert command: put dog.jpeg
[CLIENT]: Sending files...
Sending list of 47 elements
Sent packet no. 0
Server received packet!
Sent packet no. 1
Server received packet!
Sent packet no. 2
Server received packet!
Sent packet no. 3
```

4) Il server riceve la richiesta ed elabora la risposta;

```
[SERVER]: Recived PUT request from ('127.0.0.1', 59330)
packet and index number received correctly 0
packet and index number received correctly 1
packet and index number received correctly 2
packet and index number received correctly 3
```

5) Il server invia la risposta al Client, confermando che l'operazione richiesta è andata a buon fine;

```
[SERVER]: Finished reciving data from client (127.0.0.1, 59330)
[SERVER]: File dog.jpeg --> saved
[SERVER]: Response sent to (127.0.0.1, 59330)
```

6) Il Client riceve il messaggio di conferma da parte del Server;

```
[CLIENT]: Server (127.0.0.1, 10000) finished sending file!
[CLIENT]: File transfer was successful on ip: 127.0.0.1, 10000
```

7) Il Client invia una richiesta di chiusura…

```
Insert command: exit

[CLIENT]: Closing...
```

…sia il Client che il Server vengono arrestati.

```
[SERVER]: Recived EXIT request from ('127.0.0.1', 59330)
[SERVER]: Closing...
```