Elaborato per il corso

di Programmazione di Reti

2021/2022

Architettura server client UDP

Benedetta Pacilli

Matricola : 0000975296

Email: benedetta.pacilli@studio.unibo.it

Utilizzo

Per poter utilizzare le varie componenti del progetto è necessario aver installato Python, più precisamente una versione che sia uguale o superiore a Python 3.6.
Dopo di che basta clonare il repository https://github.com/benedettapacilli/UDP-client-server-architecture e andare da terminale nella cartella in cui è stato fatto il cloning.
Si apra una nuova scheda di terminale per ogni nuova componente che si vuole eseguire (ad esempio tre schede per avere il server e due client).
Per eseguire il server basta andare in src e poi in server ed eseguire il comando:

```
python3 server.py
```

Per eseguire un client basta andare in src e poi in client ed eseguire il comando:

```
python3 client.py
```

Tutti i file che vengono scambiati tra server e client vengono salvati nelle apposite cartelle *server_files* e *client_files*. Quando queste due cartelle non esistono e sono necessarie per il completamento di un'operazione vengono automaticamente create.

Descrizione

L'elaborato si compone di due script principali:

- **server.py**
- **client.py**
  e di due script librerie contenenti le funzioni necessarie:
- **server_library.py**
- **client_library.py**

server.py

A inizio script viene creato il socket e vengono impostati *localhost* e *10000*, rispettivamente come indirizzo IP e porta. Questi due dati, che formano il server address, sono subito mostrati all'avvio del server, il quale poi fa il bind per associare il server address al socket e infine, si mette in ascolto.
Per poter gestire le richieste di più client è stato importato *threading* in modo da creare un thread per ogni richiesta che si riceve.
Ogni richiesta di client viene gestita tramite la funzione *handler*: la funzione prende come argomenti l'operazione da svolgere e l'address del client che ha effettuato la richiesta; controlla se l'operazione è una di quelle ammissibili (list, get, put, exit) e in base a che operazione è viene chiamata un'apposita funzione presa da **server_library.py** (che viene importato in server.py).

server_library.py

File Python usato come libreria per il Server.
Contiene le seguenti funzioni:

- **list :**
  Viene creata una lista di stringhe contenente i nomi dei file dell'apposita cartelle *server_files* (qui vengono salvati i file che arrivano da client o che devono essere inviati).
  Come prima cosa viene inviato il numero di file presenti poi, ogni nome di file presente in questa lista viene inviato al client.

- **send :**
  Se un client richiede l'operazione *get <fileName>* il server utilizza la funzione *send* alla quale vengono passati il socket, l'address del server e l'operazione richiesta. Viene presa l'operazione, composta di due parole e con una split, si prende il secondo termine, rappresentante il nome del file richiesto. Se è presente in *server_files*, il file viene aperto in modalità di lettura e viene inviato a pacchetti di dimensioni della costante *BUFFERSIZE*. Una volta finito d'inviare il file, come ultima cosa, viene inviata la costante EOF.
  In questo modo, se quel file scelto dal client esiste già in *client_files*, questo viene sovrascritto.
- **receive :**
  La funzione receive viene chiamata quando un client sceglie l'operazione *put <fileName>*. A questa funzione vengono passati l'address del server e l'operazione del client, di quest'ultima viene presa in considerazione la seconda parola(il nome del file da inviare al server). In *server_files* viene aperto, in modalità scrittura, un file con lo stesso nome del file richiesto dal client. Finché il file non finisce (ovvero quando si legge la costante EOF), questo viene inviato a pacchetti di dimensione *BUFFERSIZE* e scritto sul file creato in *server_files*; dopo di che il file viene chiuso.
  In questo modo, se quel file scelto dal client esiste già in *server_files*, questo viene sovrascritto.
- **end_process :**
  Quando un client sceglie la funzione *exit* sia il socket che il processo del server vengono chiusi.

client.py

A inizio script viene creato il socket e vengono impostati *localhost* e *10000*, rispettivamente come indirizzo IP e porta, per creare il server address.
Viene poi chiesto d'inserire un operazione, per sapere quali sono le operazioni disponibili basta usare il comando **help**. Per ogni operazione inserita che non rientra tra quelle valide viene mostrato un messaggio di errore.

Per ogni operazione valida scelta viene chiamata una funzione di **client_library**, importata in client.py

client_library.py

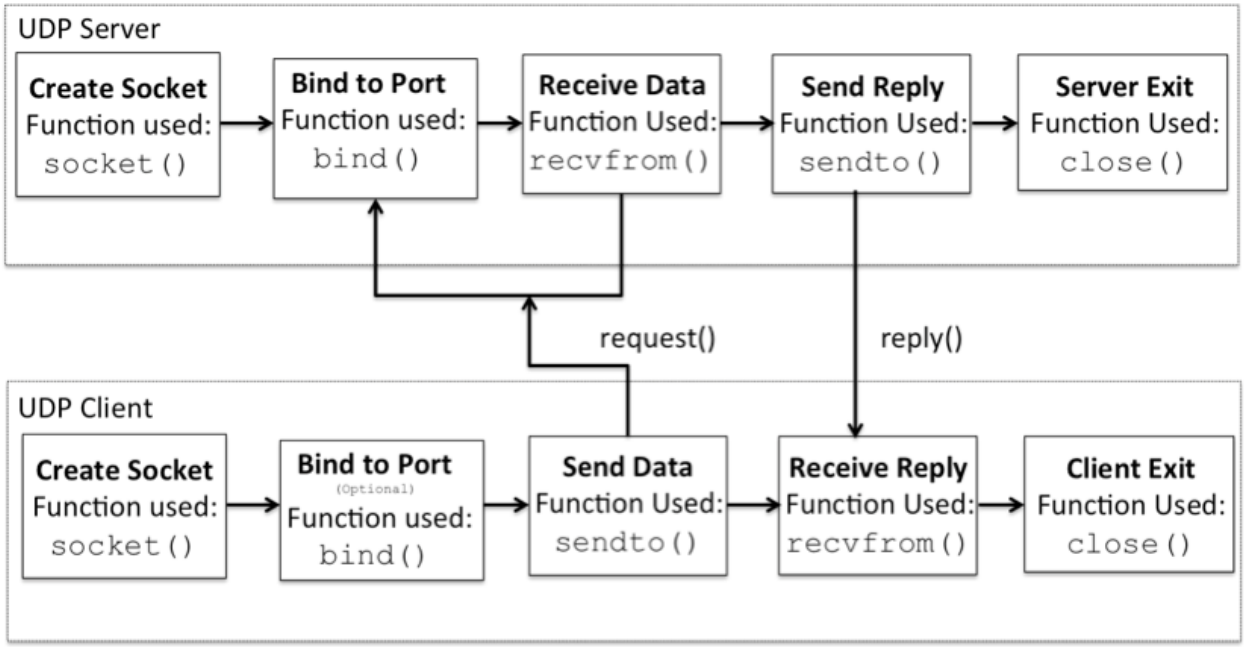
File Python usato come libreria per i Client.
Contiene le seguenti funzioni:

- **list:**
  Come prima cosa dice al server che l'operazione richiesta è *list*. Poi il client riceve il numero di file presenti in *server_files*. Se il numero è zero, viene mostrato un messaggio che indica la non presenza di file sul server.
  In caso ci siano file: per ognuno viene fatta una *receive* per ottenere il nome del file, che a questo viene mostrato a schermo.
- **get:**
  Dice al server che l'operazione richiesta è la *get* inviando anche il nome del file richiesto. Se il file non è stato trovato tra quelli del server, viene mostrato un messaggio che avvisa la non presenza del file. Se invece il file è presente, in *client_files* viene aperto, in modalità scrittura, un file con lo stesso nome del file richiesto dal client. Finché il file non finisce (ovvero quando si legge la costante EOF), questo viene inviato a pacchetti di dimensione *BUFFERSIZE* e scritto sul file creato in *client_files*; dopo di che il file viene chiuso.
  In questo modo, se quel file scelto dal client esiste già in *client_files* , questo viene sovrascritto.
- **put:**
  Prende la seconda parola del'operazione e la salva come nome del file da inviare al server.
  Se il file è presente in *client_files*: viene mandata l'operazione richiesta al server, viene aperto il file (specificano nell'operazione) in modalità di lettura e viene inviato a pacchetti di dimensioni della costante *BUFFERSIZE*. Una volta finito d'inviare il file, come ultima cosa, viene inviata la costante EOF.
  Se invece il file non è presente viene mostrato un messaggio che avvisa il client.

In questo modo, se quel file scelto dal client esiste già in *server_files* , questo viene sovrascritto.

- **end_process:**
  Quando un client sceglie la funzione *exit* sia il socket che il processo del client vengono chiusi.

La seguente immagine mostra come il server e i client comunicano.



Uso dei thread

In server viene importato *threading*.
Per ogni client che fa una richiesta al server viene creato un nuovo *thread* (come target del thread viene specificata la funzione handler, definita in server.py e come args vengono passati l'operazione richiesta e il server address), successivamente si fa iniziare il thread e si fa il *join*. Ogni thread chiamato viene bloccato e deve aspettare che termini il precedente thread il cui join è stato chiamato.
Questo fa in modo che non vi siano letture/scritture contemporanee su file da parte di diversi client.
La funzione *handler* serve a gestire l'operazione per la quale è stato creato il thread. Solo l'operazione exit non viene gestita nell'handler. Ma direttamente nel loop principale del server. Infatti, in caso l'operazione sia *exit*, non viene creato alcun thread ma si chiama direttamente l'apposita funzione *end_process* che si trova in *server_library.py*.