

# Programmazione di Reti 2021/2022 - Ingegneria e Scienze informatiche

## Relazione Progetto di Reti - client-server-udp- protocol

Salvatore Antonio Addimando - 0000970539

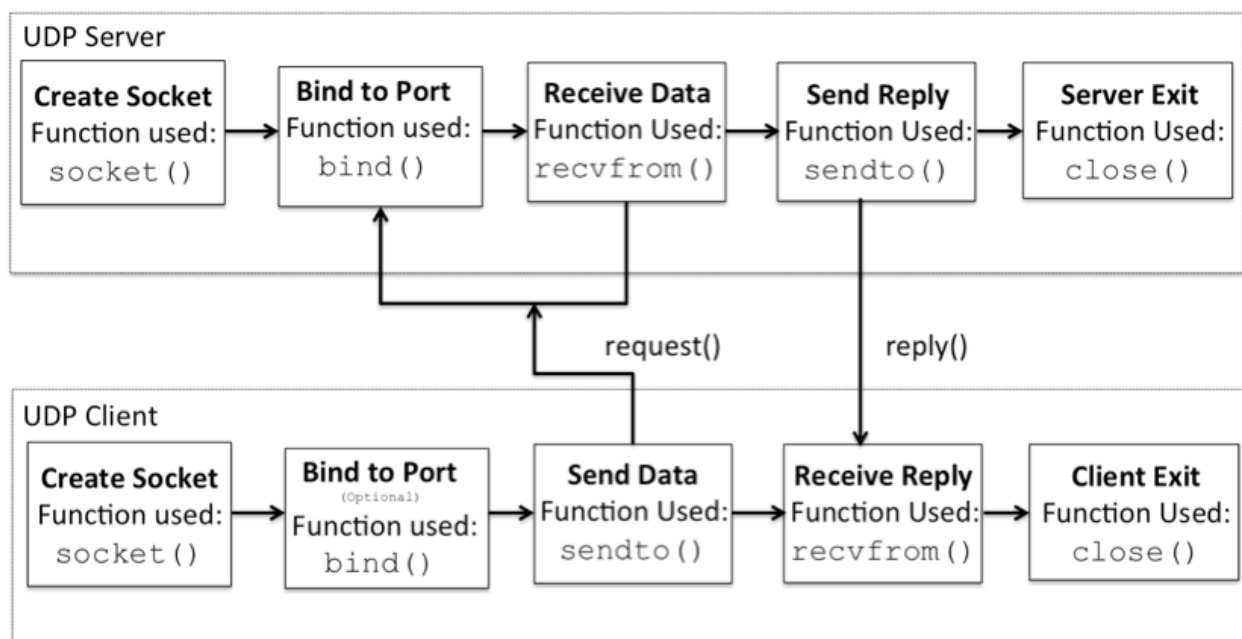
### Indice

- Introduzione
- Descrizione
  - Descrizione delle funzioni

### Introduzione

Questa relazione ha lo scopo di descrivere e spiegare le scelte effettuate, volte alla realizzazione della seconda traccia dell'elaborato del corso di Programmazione di Reti. La seconda traccia consiste nella realizzazione di un server-client UDP.

### Descrizione



La mia implementazione di questo progetto si sviluppa in due directory principali: **client** e **server**, che racchiudono rispettivamente l'applicativo lato client e l'applicativo lato server.

In ognuna di queste directory troviamo due script in python: uno che contiene le operazioni principali del client (o del server se nell'altra directory) e uno che è lo script eseguibile che permette di comunicare col server (o col client).

Per ogni client che tenta la connessione il server crea un thread apposito. Su tutti i thread viene chiamata `join()` e in questo modo si crea una sorta di coda di client a cui rispondere, così evitando anche i problemi di concorrenza sulla scrittura e lettura sui/dei file del server.

## Descrizione delle funzioni (**list**, **get**, **put**, **exit**)

### **list**

Quando il client invia il comando **list** al server, quest'ultimo recupera la lista di file presenti nella directory `./files` e manda prima il numero di file trovati, che viene salvato in un'apposita variabile nel client, e poi manda uno alla volta i nomi dei file mentre il client riceve tante volte quante il numero di file trovati e ad ogni nome ricevuto lo stampa a schermo.

### **get**

Il client può inviare il comando **get** seguito dal nome di un file che si vuole ricevere; il server controlla innanzitutto se il file è presente o no e se non lo è lo comunica al client, se è presente il client apre un file omonimo in scrittura e continua a ricevere dati di lunghezza `BUFFER_SIZE` (che non indica altro che la grandezza in byte di un pacchetto singolo) scrivendoli man mano sul file aperto in precedenza fino a quando il dato mandato è una costante chiamata `EOF` inizializzata appositamente per essere distinguibile da qualsiasi riga dell'effettivo file, in modo da sapere esattamente quando è stato completato il trasferimento.

Alla fine di questo procedimento il file viene chiuso e viene fornito un feedback sul completamento dell'operazione al client.

### **put**

Il client può inviare il comando **put** seguito dal nome di un file che si vuole inviare; il client controlla se il file è presente o no e se non lo è mostra un messaggio a schermo per informare l'utente. Se invece è presente invia il comando al server e apre il file specificato in lettura. Legge una quantità di dati lunga `BUFFER_SIZE` byte, dal file, alla volta e la invia al server fino a quando la porzione di file letta è null inviando quindi la costante `EOF` per indicare la fine del file. Nel frattempo il server continua a ricevere dati di lunghezza `BUFFER_SIZE` e a scriverli sul file fino alla ricezione della costante

EOF. Alla fine della procedura il file viene chiuso sia sul client che sul server e il client mostra a schermo un feedback sulla conclusione con successo dell'operazione.

## **exit**

All'invio del comando **exit** il server chiude il socket e interrompe l'esecuzione dello script e lo stesso accade lato client.