Relazione progetto Programmazione di Reti 2021-22

Studente: Emanuele Borghini, emanuele.borghini@studio.unibo.it

Matricola: 0000988152

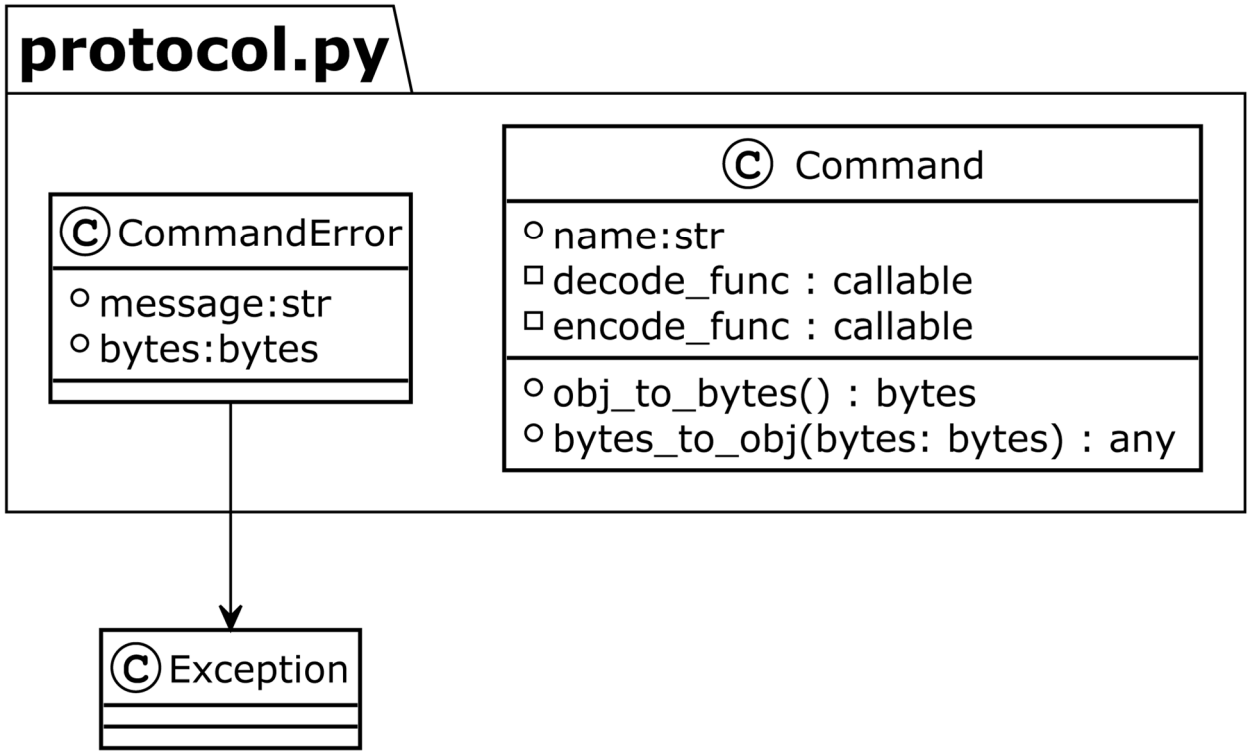
Traccia: 2 “Architettura client-server UDP per trasferimento file”

Repository: https://github.com/Borgotto/udp-file-transfer

(le istruzioni per l’uso si trovano nel file README.md)

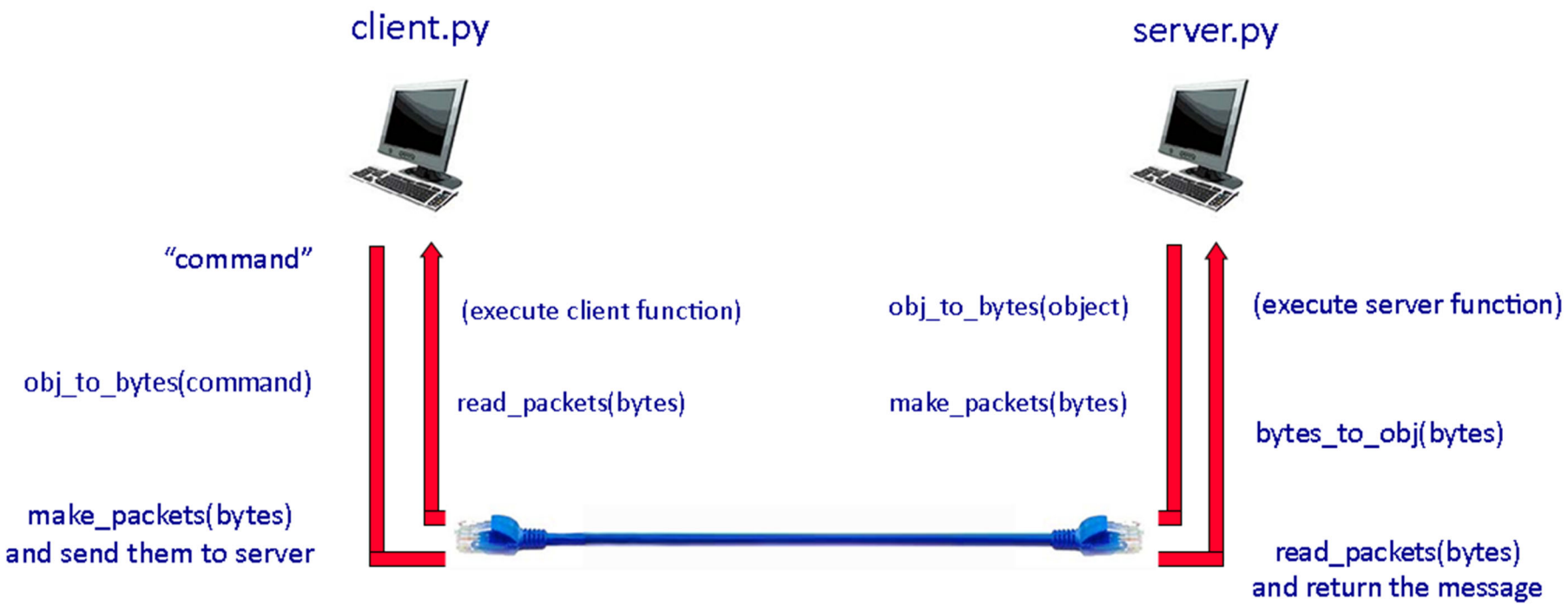
Overview

Diagramma UML



Overview

Flow diagram:



Progettazione

L'obbiettivo principale era quello di separare l'implementazione **dell'architetture client-server** dalle funzionalità di essi, per questo **ho definito un file** *protocol.py* che contiene tutte le regole e specifiche da seguire per permettere lo scambio di messaggi.

Il **client** (*client.py*) e **server** (*server.py*) seguono implementazioni piuttosto basiche di *socket UDP* sfruttando le librerie di python3, entrambi hanno dei ***while loop*** dove ricevono pacchetti, eseguono i comandi correlati ai dati da ricevere/mandare, entrambi eseguono su **un solo thread principale** in quanto il client non necessita di più input allo stesso tempo, mentre il server va in contro a problemi di input-output in caso contrario (scrittura/lettura file richiesta da due client nello stesso istante), nonostante ciò il protocollo scritto fornisce il supporto per l'implementazione di un server capace di gestire ***n*** numero di client simultaneamente.

Come si può osservare dal diagramma di flusso della pagina precedente, il percorso di un pacchetto parte dal client che digita un comando e parametri validi, il comando di tipo ***protocol.Command*** definisce le funzioni che andranno ad eseguire prima di inviare i ***byte*** e una volta ricevuti per riconvertire la risposta (in byte) a un oggetto, dopo di ché i ***byte*** vengono impacchettati secondo ***protocol.make_packets***, una volta arrivati al server viene ripetuto lo stesso processo all'inverso, i pacchetti vengono letti, il comando esegue una funzione e la risposta viene inviata al client.

Strutture Dati

Il client non fa uso di particolari strutture dati, i comandi inseriti dall'utente sono semplici stringhe e i byte da inviare sono bytes, entrambi tipi *built-in* di python.

Il server invece per concedere a più client di connettersi allo stesso tempo, fa uso di un **dizionario** con **chiave** di tipo *tuple[str,int]* che identifica un client tramite *[ip,port]* e **valore** *list[bytes]* contenente tutti i byte ricevuti dal client, la coppia chiave-valore viene poi cancellata al termine dello scambio di messaggi.

La parte più importante delle **strutture dati e costanti** vengono definite in *protocol.py*;

il file contiene:

tutte le costanti **necessarie per settare i socket del client e server** e per definire il formato dei pacchetti;

la classe *Command* permette facilmente di creare nuovi comandi e risposte per il client e server;

la classe *CommandError* è un'eccezione che permette di inviare casi particolari di comandi per gestire eventuali problemi riscontrati nello scambio di messaggi (es. file non esiste);

Infine, due dizionari: *commands* e *responses*, che contengono le istanze della classe Command, rispettivamente per i comandi inviati dal client e le risposte inviate dal server.