Matricola n°

Email: riccardo.tassinari9@studio.unibo.it

TRACCIA 2: architettura client-server UDP per trasferimento file.

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 permette**:

● Connessione senza autenticazione.

● Visualizzazione dei file disponibili per il trasferimento.

● Download di un file selezionato sul client.

● Upload di un file selezionato sul server.

**Il protocollo di comunicazione prevede lo scambio** **di** :

● Messaggi di comando o di errore rilevato: vengono inviati dal client al server per richiedere l’esecuzione delle diverse operazioni o per annullare l’operazione.

● Messaggi di risposta: vengono inviati dal server al client in risposta ad un comando con l’esito dell’operazione.

**Funzionalità server:**

● L’invio del messaggio di risposta al comando list al client richiedente contenente la file list, ovvero la lista dei nomi dei file disponibili per la condivisione, con la rispettiva **dimensione in byte**. Nel caso non fosse presente alcun file, viene restituito un messaggio opportuno.

● 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. Nel caso **il file esistesse** già, viene restituito al server un messaggio opportuno.

● Gestione degli errori.

**Funzionalità client:**

● 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 (es. file richiesto non esiste).

● 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.

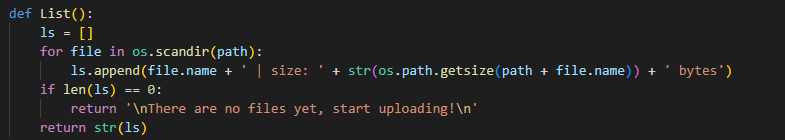
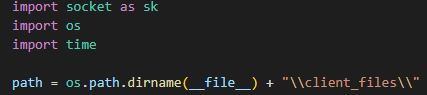
● Gestione dell’input per **prevedere possibili errori di sintassi**, da comunicare preventivamente al server.

● Comando **close** per chiudere il socket e terminare l’applicazione.

**Sviluppo:**

Per lo sviluppo dell’applicazione ho importato alcuni moduli necessari, e altri non necessari, alla corretta implementazione:

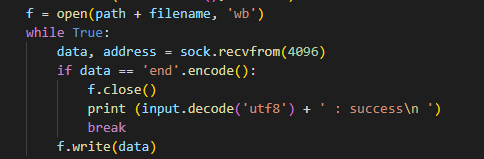
● **Modulo** **os:** ho utilizzato il modulo os per gestire tutte le operazioni di ricerca file, dimensioni file, esistenza file e ricerca repository.

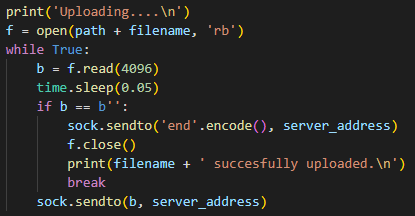


● **Modulo** **time:** mi sono avvalso del modulo time per motivi puramente estetici nell’output.

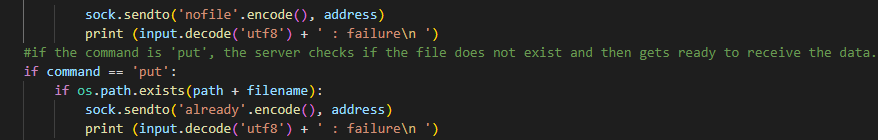
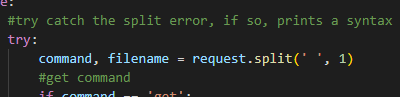


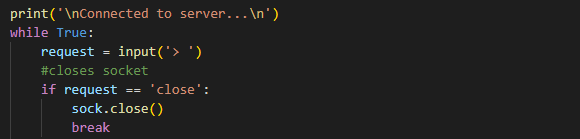
● Ho utilizzato la **scrittura e lettura dei file in binary mode** così da poter inviare e scrivere blocchi di byte in un file fino al termine dell’operazione di download/upload, che viene annunciata dall’invio di un opportuno messaggio al ricevente del file.

scrittura

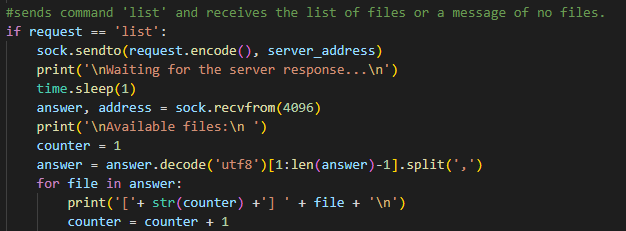
Lettura

● Ho fatto in modo di **gestire tutti i possibili errori** di sintassi, mancanza di file, comandi, senza però interrompere la connessione client-server, che può essere interrotta con l’immissione del comando **close,** che chiude il client socket.





● Mi sono poi concentrato sul **lato estetico** dell’applicazione cercando di dare un ordine a tutta la parte riguardante l’output, sia da parte del client sia da parte del server. Per fare ciò ho lavorato con le stringhe e gli output, aggiungendo opportunamente anche dei time.sleep() per far apparire più fluide e meno caotiche le operazioni.



**Modelli:**