

Relazione del Progetto di Programmazione di Reti

Traccia N°2

In questo progetto ho realizzato un'architettura client-server UDP che permette il trasferimento di file.

Guida all'utilizzo del programma

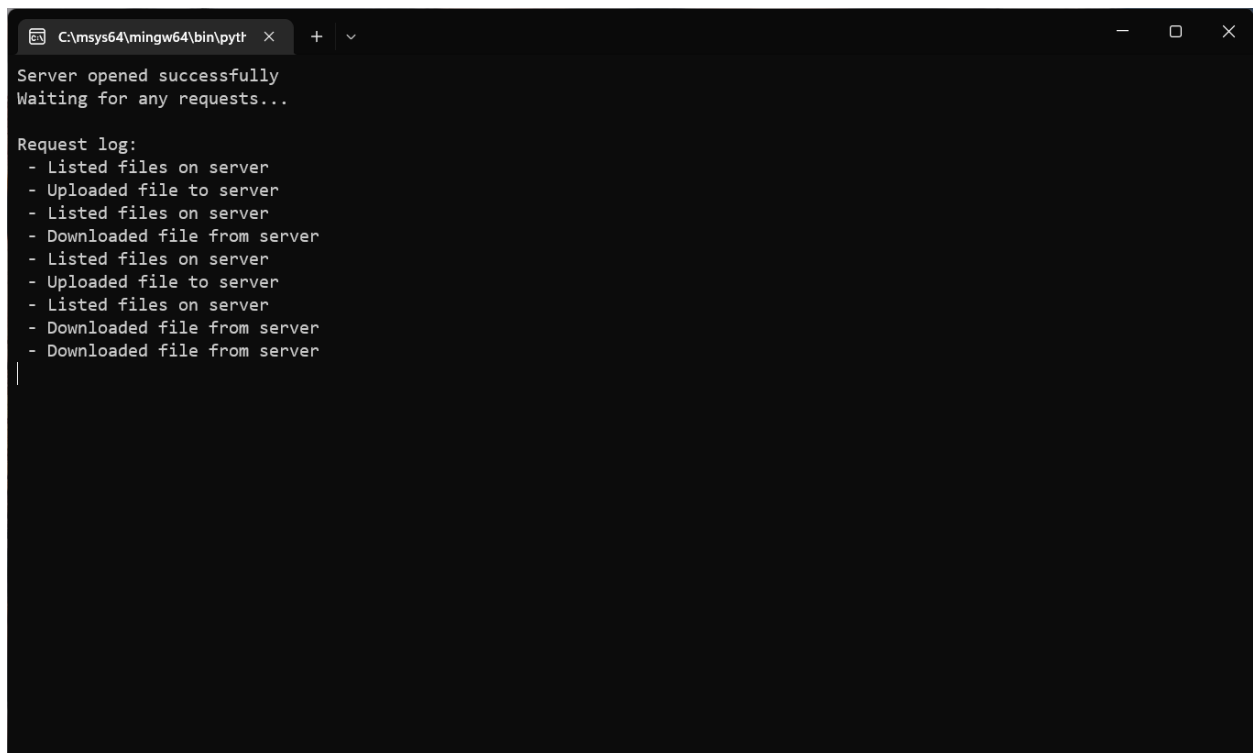
Lanciare il server e il client tramite comando del terminale:

```
python Server.py
```

```
python Client.py
```

o, equivalentemente, avviando `Launch.bat`.

Il server presenta un'interfaccia esclusivamente a linea di comando, in cui vengono visualizzati messaggi a schermo relativi alla corretta apertura del server e ai comandi che vengono richiesti al server da parte del client.



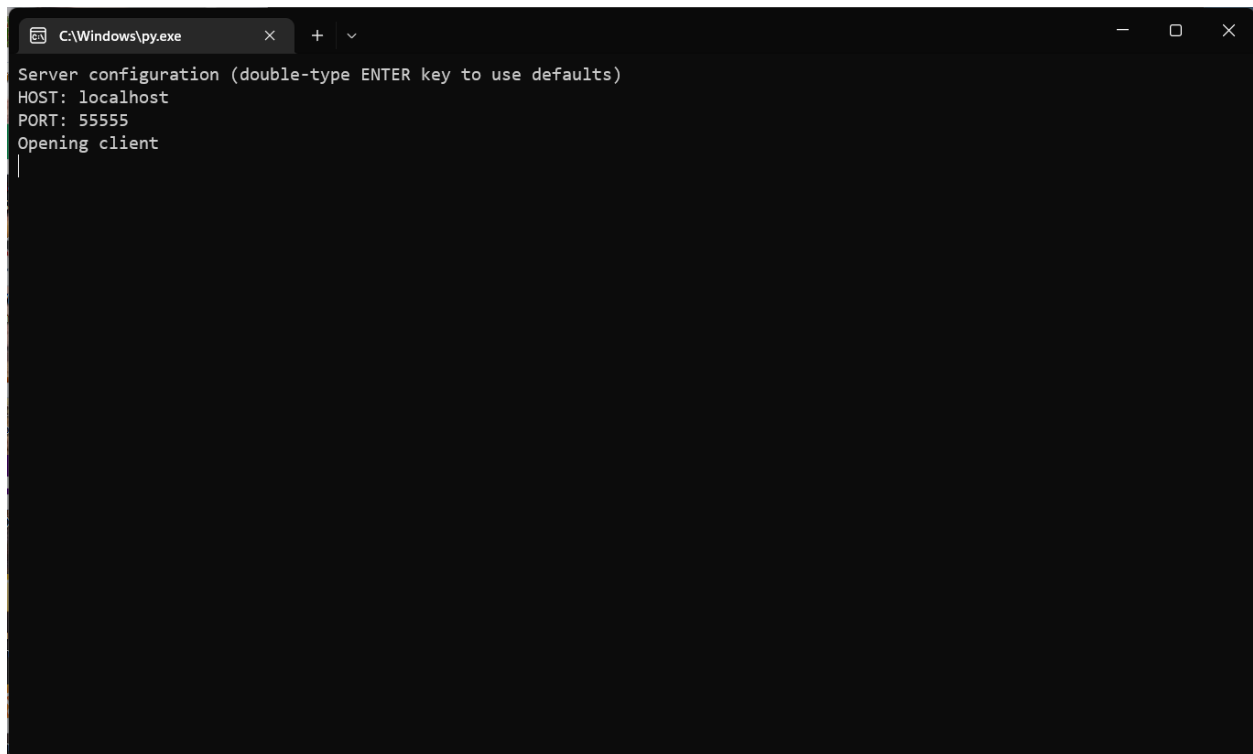
```
C:\msys64\mingw64\bin\pyth x + v
Server opened successfully
Waiting for any requests...

Request log:
- Listed files on server
- Uploaded file to server
- Listed files on server
- Downloaded file from server
- Listed files on server
- Uploaded file to server
- Listed files on server
- Downloaded file from server
- Downloaded file from server
```

Il client presenta un'interfaccia grafica, sviluppata con Tkinter, che permette all'utente di effettuare tre tipi diversi di richieste:

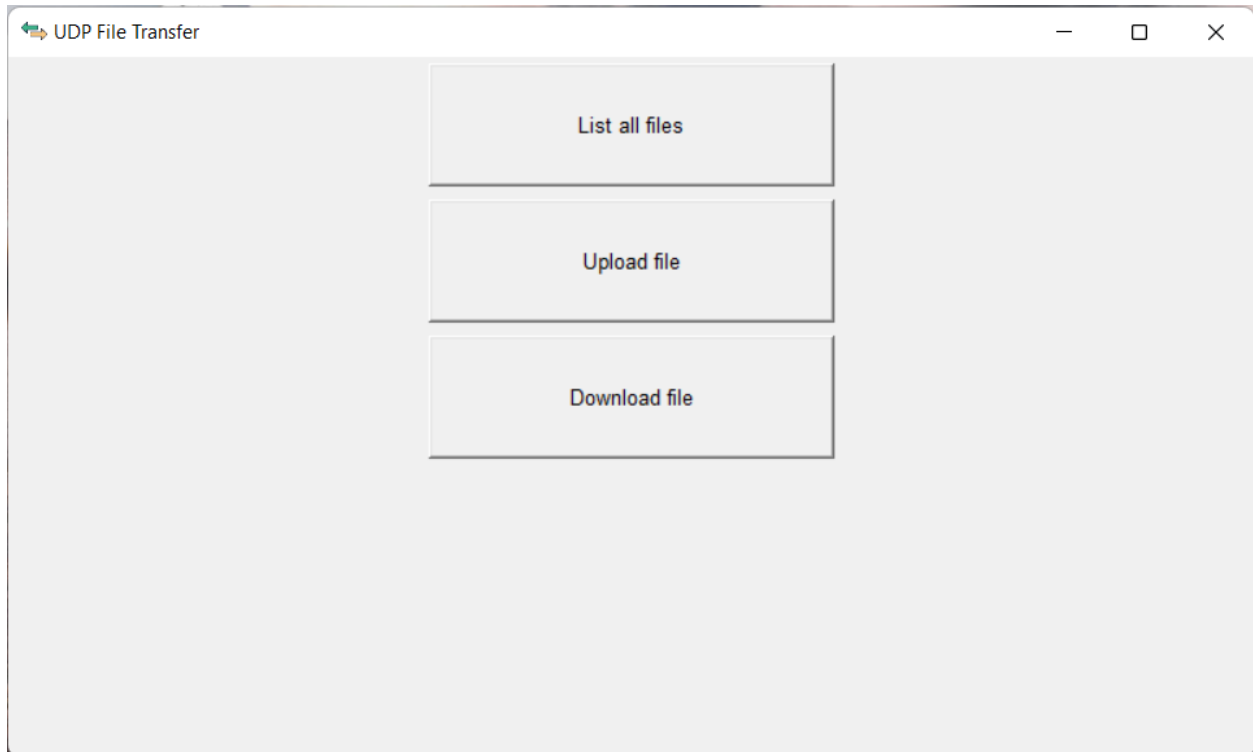
- Visualizzare la lista dei nomi dei file disponibili sul server
- Scaricare un file presente sul server
- Caricare un file locale sul server

All'avvio del client compare un messaggio su linea di comando che richiede all'utente di inserire prima l'indirizzo del server e, successivamente, la porta su cui si effettua la connessione. Per inserire i valori di default, basta premere due volte il tasto invio. Dopodiché, verrà mostrata la GUI sopra citata.

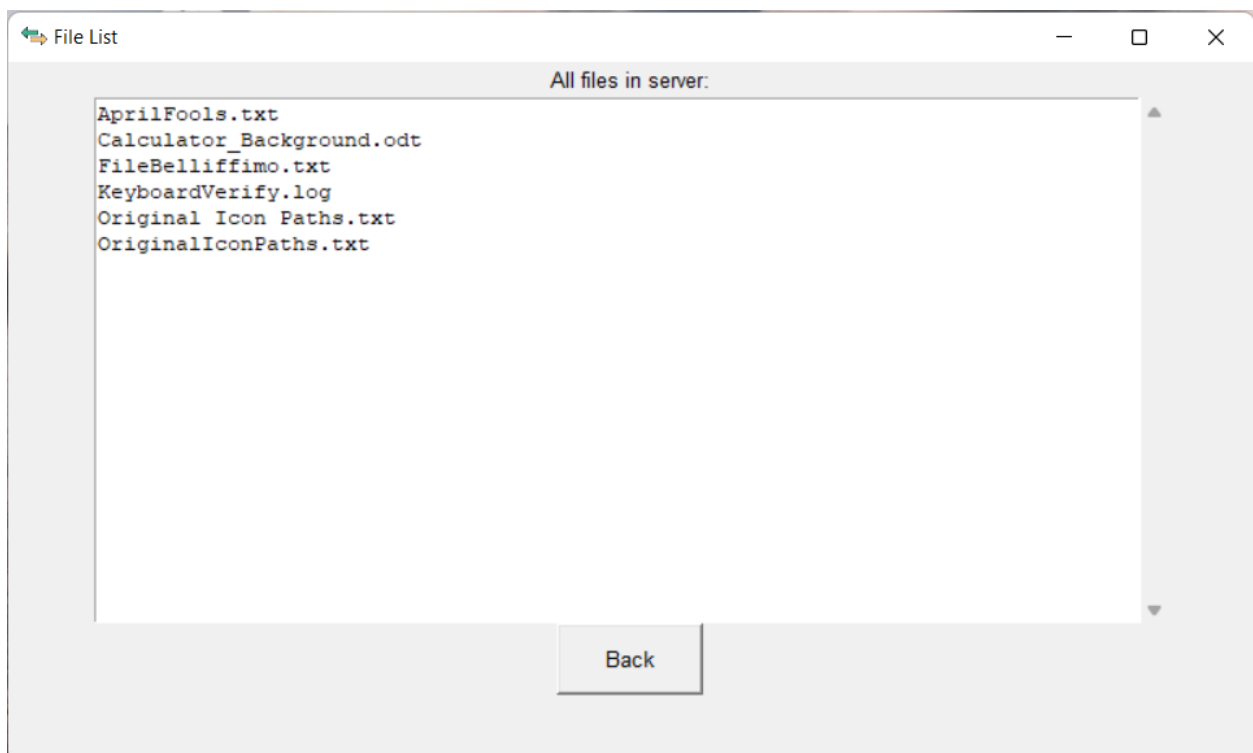
A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\py.exe' with standard window controls. The command prompt displays the following text: 'Server configuration (double-type ENTER key to use defaults)', 'HOST: localhost', 'PORT: 55555', and 'Opening client'. A vertical cursor is visible on the line following 'Opening client'.

```
C:\Windows\py.exe
Server configuration (double-type ENTER key to use defaults)
HOST: localhost
PORT: 55555
Opening client
|
```

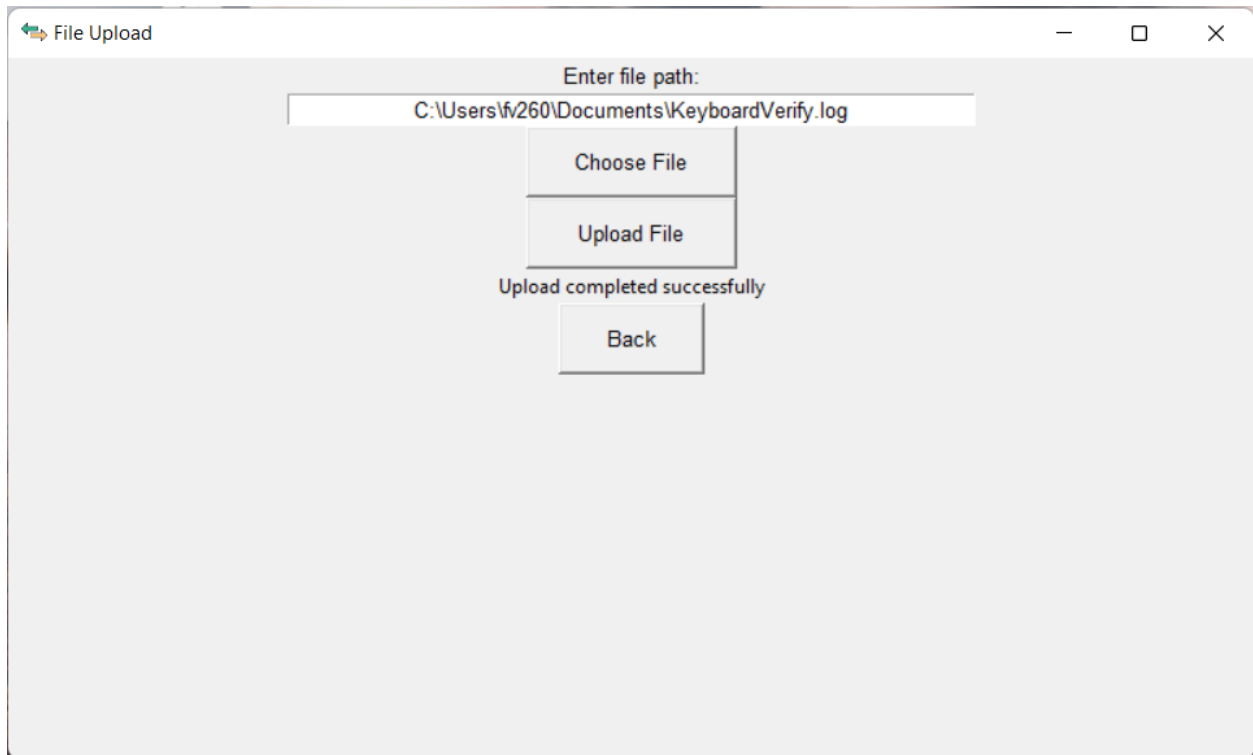
L'interfaccia del menù principale contiene tre bottoni, uno per ogni tipologia di richiesta.



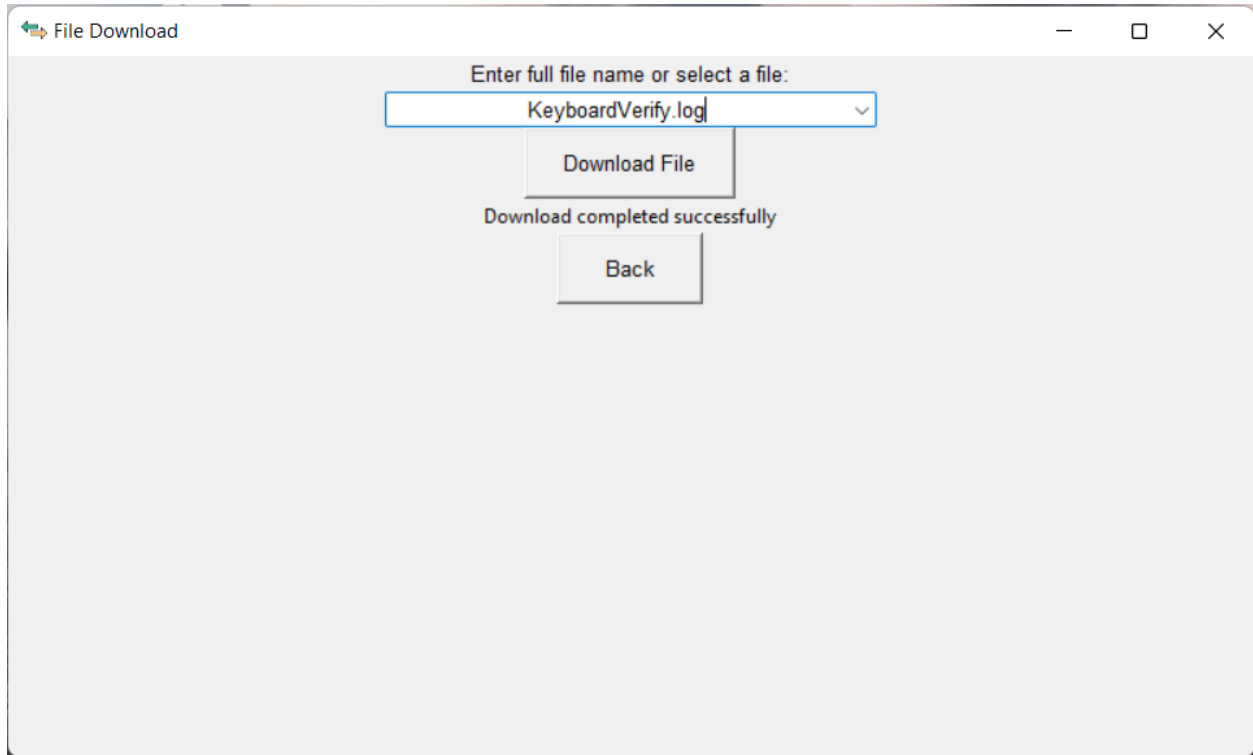
Selezionando il primo pulsante, comparirà una schermata in cui verrà mostrata la lista dei file disponibili sul server.



Selezionando il secondo pulsante, comparirà una schermata che permetterà di caricare un file locale sul server. Il file può essere selezionato tramite l'apposito bottone, o inserendo manualmente il percorso nel relativo campo. In seguito, potremo caricare il file cliccando sul pulsante "Upload" e l'esito dell'operazione verrà mostrato con un messaggio apposito.



Selezionando il terzo pulsante, comparirà una schermata che permetterà di scaricare un file presente sul server. Per selezionare il file possiamo utilizzare l'apposito menù a tendina, o specificare manualmente il nome del file nel relativo campo. Cliccando poi sul pulsante "Download" verrà scaricato il file e l'esito dell'operazione sarà mostrato mediante un apposito messaggio.



Programmazione del Client

All'avvio il client crea il socket, memorizza l'indirizzo del server, crea la GUI e aspetta un input dalla stessa. Presenta un singolo thread.

Ha 12 funzioni, che svolgono diversi compiti all'interno del programma:

- Gestione delle richieste: vengono richiamate cliccando sui bottoni di Download e Upload all'interno delle rispettive schermate
 - **get(file_name, message_label)**
Invia al server il messaggio "2", seguito dal nome del file che vogliamo scaricare, per indicare la richiesta da eseguire. Poi, riceve la risposta e si occupa di gestire eventuali errori e di salvare in locale il file ricevuto. Se il contenuto del file avesse una dimensione maggiore del buffer dei messaggi client-server, verrà inviato in più pacchetti, indicando opportunamente tale variazione mediante il primo byte: "1" indica che è avvenuto un errore, "2" indica la presenza di ulteriori pacchetti, "0" indica il trasferimento senza errori dell'ultimo pacchetto.
 - **put(file_path, message_label)**
Apri il file locale richiesto e, se l'operazione dà esito positivo, invia al server il messaggio "3", seguito dal nome del file da caricare. In seguito, invia al server uno o più pacchetti che contengono il contenuto del file, indicando come primo byte "2" se sono presenti ulteriori pacchetti e "0" se viene inviato l'ultimo pacchetto senza errori.
- Gestione del cambio del Frame nella GUI
 - **list_handler()**
Invia al server il messaggio "1", indicando la richiesta da eseguire (List Files), riceve la risposta con la lista dei file e la visualizza in un nuovo frame.
 - **get_handler()**
Mostra l'interfaccia utente per effettuare il download in un nuovo frame.
 - **put_handler()**
Mostra la UI per effettuare l'upload, sempre in un nuovo frame.
 - **create_menu_frame()**
Mostra la UI del menù principale, in un nuovo frame.

- **back(frame)**
Torna al menù principale e cancella il vecchio frame.
- **raise_frame(frame)**
Mostra il frame selezionato.
- Esecuzione del trasferimento dei dati vero e proprio
 - **send(message)**
Invia al server il messaggio message
 - **receive()**
Gestisce la ricezione dei messaggi inviati dal server. Si occupa di parte del protocollo di comunicazione client-server, controllando il primo byte del messaggio ricevuto: "0" indica che la comunicazione è avvenuta senza errori, mentre "1" e "2" indicano che sono avvenuti errori nella comunicazione.
- Funzioni miscellanee
 - **on_close()**
Chiude la GUI e il socket alla chiusura della finestra o del programma.
 - **get_file_list()**
Richiede al server la lista dei file disponibili sullo stesso e ritorna la stessa lista in un formato più fruibile (da lista di caratteri a lista di stringhe aventi un significato).

Programmazione del Server

All'avvio, il server crea il socket e si mette in attesa di richieste invocando la funzione `wait_for_requests()`. Presenta un singolo thread.

Ha 5 funzioni, che svolgono diversi compiti all'interno del programma:

- **Gestione delle richieste**
 - **`list_files()`**
Restituisce la lista dei file presenti.
 - **`get_file(file_name, client_address)`**
Apre il file avente il nome ricevuto. Se l'apertura del file dovesse fallire, viene inviato il messaggio "1" al client, segnalando l'errore. Se invece avviene con successo, il server invia uno o più pacchetti contenenti il contenuto del file aperto, inserendo come primo byte "2" se sono presenti ulteriori pacchetti, "1" se è avvenuto un errore e "0" se viene inviato l'ultimo pacchetto senza aver riscontrato errori.
 - **`put_file(file_name, client_address)`**
Crea e scrive il file avente il nome ricevuto. Attende l'arrivo del contenuto del file da parte del client e, se il messaggio contiene "2" come primo byte, il server aspetta ulteriori pacchetti, mentre se contiene "0" la scrittura termina, in quanto era l'ultimo pacchetto da scrivere.
- **Ricezione delle richieste**
 - **`wait_for_requests()`**
È la funzione principale, che crea un ciclo infinito in cui aspetta di ricevere messaggi dal client e, alla loro ricezione, chiama la relativa funzione per gestire la richiesta ricevuta. Gestisce parte del protocollo di comunicazione client-server, prefissando il messaggio da inviare col byte "0" se la gestione della richiesta è avvenuta correttamente, "1" se la richiesta ricevuta non è valida, "2" se è avvenuto un errore nella comunicazione.
 - **`receive(client_address)`**
È utilizzata dalle funzioni `get_file()` e `put_file()` per aspettare i messaggi dal client. Se il client impiega troppo tempo per inviare il messaggio, la funzione va in timeout, rendendo il server nuovamente disponibile ad accettare ulteriori richieste.

Protocollo utilizzato

Per una maggiore chiarezza, riportiamo nuovamente un riassunto del funzionamento del protocollo.

Protocollo generale:

Il client invia al server le richieste, specificando i numeri "1", "2", "3", seguiti dal nome del file da scaricare nel caso "2", e dal nome del file da caricare nel caso "3".

Il server riceve le richieste e risponde segnalando eventuali errori, tramite l'invio dei numeri:

- "0": messaggio ricevuto correttamente. La richiesta viene gestita e il restante messaggio di risposta viene accodato allo "0"
- "1": richiesta non valida
- "2": errore di comunicazione

Protocollo specifico per ogni richiesta:

- **LIST**: il server risponde al client accodando la stringa contenente la lista dei file presenti sul server.
- **GET**: il server risponde al client accodando un messaggio composto da un numero:
 - "0": ultimo pacchetto, nessun errore,
 - "1": errore,
 - "2": presenza di ulteriori pacchetti,

seguito dal contenuto del file o da parte di esso se è di dimensione maggiore del buffer. Se sono presenti più pacchetti ("2"), ad ogni conferma di avvenuta ricezione del messaggio da parte del client, il protocollo si ripete, fintanto che il server arriva ad inviare "0" oppure "1" (es: la connessione va in timeout).

- **PUT**: il server risponde al client solo con lo "0", per confermare l'avvenuta ricezione del file o da parte di esso se è di dimensione maggiore del buffer. Il server è poi tenuto a rispondere accodando un messaggio contenente un numero:

- "0": nessun errore,
- "1": errore,

ad ogni ricezione di pacchetto (almeno uno).

Francesco Valentini
Matricola: 0000970197
Email: francesco.valentin11@studio.unibo.it