Matricola n° 0000970372

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.

```python
import socket as sk
import os
import time

path = os.path.dirname(__file__) + "\\client_files\\"
def List():
    ls = []
    for file in os.scandir(path):
        ls.append(file.name + ' | size: ' + str(os.path.getsize(path + file.name)) + ' bytes')
    if len(ls) == 0:
        return '\nThere are no files yet, start uploading!\n'
    return str(ls)
```

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

```python
sock.sendto(request.encode(), server_address)
print('\nWaiting for the server response...\n')
time.sleep(1)
```

● 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.

```python
    f = open(path + filename, 'wb')
    while True:
        data, address = sock.recvfrom(4096)
        if data == 'end'.encode():
            f.close()
            print (input.decode('utf8') + ' : success\n ')
            break
        f.write(data)
```

scrittura

```python
print('Uploading....\n')
f = open(path + filename, 'rb')
while True:
    b = f.read(4096)
    time.sleep(0.05)
    if b == b'':
        sock.sendto('end'.encode(), server_address)
        f.close()
        print(filename + ' succesfully uploaded.\n')
        break
    sock.sendto(b, server_address)
```

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.

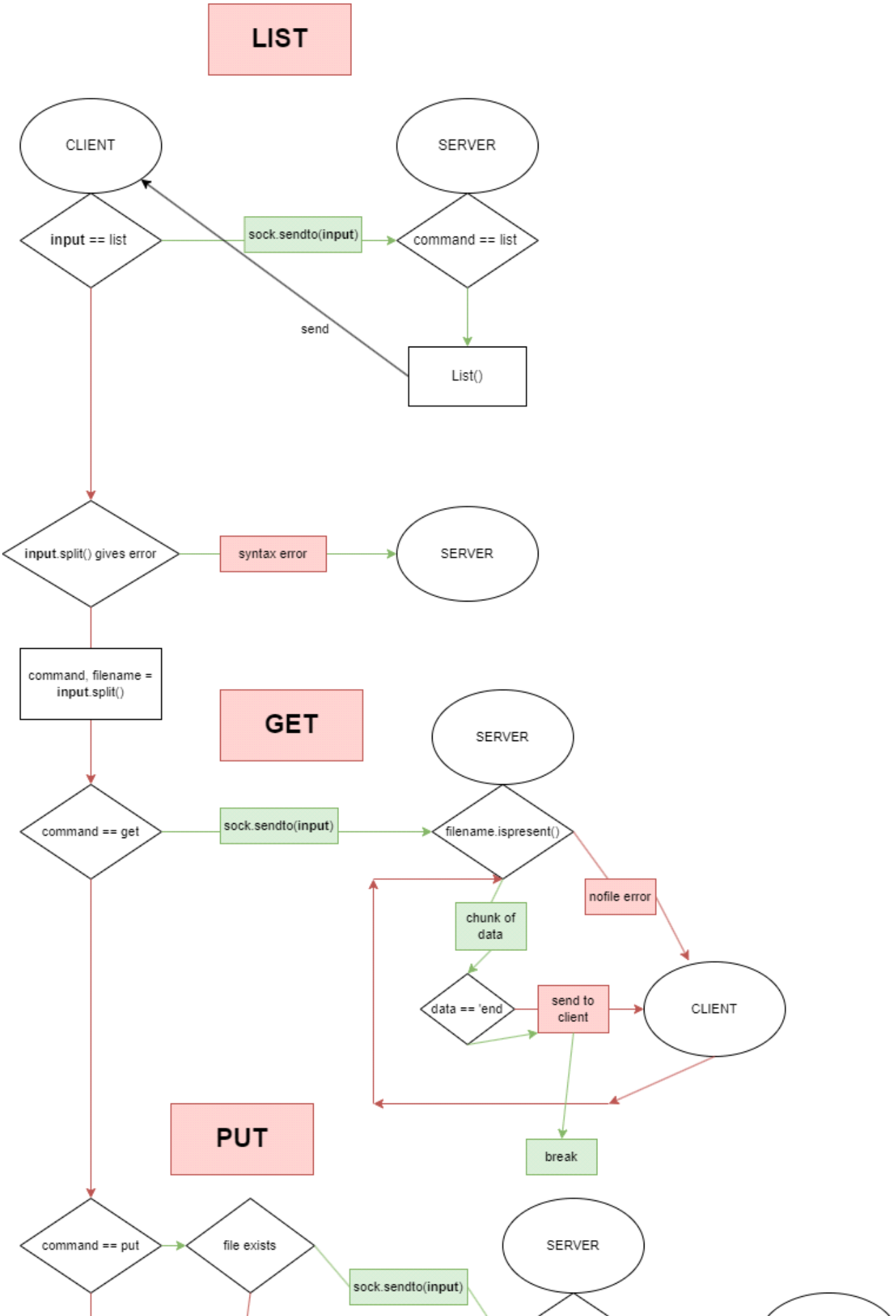
```
#try catch the split error, if so, prints a syntax
try:
    command, filename = request.split(' ', 1)
    #get command
    if command == 'get':
        sock.sendto('nofile'.encode(), address)
        print (input.decode('utf8') + ' : failure\n ')
 #if the command is 'put', the server checks if the file does not exist and then gets ready to receive the data.
 if command == 'put':
    if os.path.exists(path + filename):
        sock.sendto('already'.encode(), address)
        print (input.decode('utf8') + ' : failure\n ')
```

```
print('\nConnected to server...\n')
while True:
    request = input('> ')
    #closes socket
    if request == 'close':
        sock.close()
        break
```

● 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.

```
#sends command 'list' and receives the list of files or a message of no files.
if request == 'list':
    sock.sendto(request.encode(), server_address)
    print('\nWaiting for the server response...\n')
    time.sleep(1)
    answer, address = sock.recvfrom(4096)
    print('\nAvailable files:\n ')
    counter = 1
    answer = answer.decode('utf8')[1:len(answer)-1].split(',')
    for file in answer:
        print('['+ str(counter) +'] ' + file + '\n')
        counter = counter + 1
```

Modelli:


LIST
CLIENT
SERVER
input == list
sock.sendto(input)
command == list
send
List()
input.split() gives error
syntax error
SERVER
command, filename = input.split()
GET
SERVER
command == get
sock.sendto(input)
filename.ispresent()
nofile error
chunk of data
data == 'end
send to client
CLIENT
break
PUT
command == put
file exists
SERVER
sock.sendto(input)