Relazione

Studente: Simone Iasagni.

Matricola: 0000998742.

Email: simone.lasagni@studio.unibo.it .

▼ Apice

- Descrizione delle scelte concettuali effettuate.
- Descrizione del funzionamento del server.
- Descrizione del funzionamento del client.

Scopo del progetto - Traccia 2

Il progetto consiste nel implementare un applicazione in Python che gestisca il trasferimento di file di un client-server attraverso il protocollo UDP e il servizio di rete senza conessione.
I punti principali sono di avere una connessione senza autenticazione, di far visualizzare al client i file presenti sul server, il poter scaricare un file presente sul server e la posibilità di poter caricarne uno sul server.

In particolare il server deve rispondere con la lista dei file presenti sul server alla chiamata list del client, un messaggio di risposta con il file richiesto da scaricare alla chiamata del comando get oppure un messaggio d'errore se il file richiesto non è presente e la ricezione e lo storage di un file dal client dopo la chiamata al comando put.

Il client deve richiedere la lista dei file disponibili attraverso il comando list, ricevere e storare un file dal server o gestire l'eventuale errore dopo la chiamata get e infine inviare un file presente sul client al server alla chiamata del comando put.

Scelte concettuali

Come primo approccio ho pensato di manipolare meno possibile i pacchetti, senza l'utilizzo di un JSON o trasformarli in liste per poi in binario ed inviarli, la mia idea principale si basava sul fatto che avessi pensato solo a lavorare su file di testo, quindi aprivo un file TXT leggevo un numero presatibilito di parole dopo di che le inviavo attrverso il comando *encode*, una volta arrivato il pacchetto mi bastava usare *decode* e avevo la mia stringa pronta per essere scritta in un nuovo file, effettivamente utilizzando solo file TXT il programma funzionava bene.

Ho riscontrato problemi una volta che ho provato ad inviare una foto, il file che mi risultava dopo l'invio e la scrittura dei pacchetti o era danneggiatto ed impossibile da aprire o l'immmagine era completamente nera, in quel momento ho rivisto completamente il mio approccio ed ho pensato ad utlizzare direttamente il binario.

```
file = open("testo.txt", 'rb')
```

Potevo aprire e leggere tranquillamente un file direttamente in binario, dopodichè il suo invio e la sua scrittura era altrettanto semplici visto che l'invio dei pacchetti è in binario non dovevo nemmeno piu usare i comandi per trasformare le stringhe prima e dopo l'invio, ma anche in questo caso ho riscontrato alcuni prolemi con le immagini e i video, molti frame non erano giusti cambiando i colori originali oppure il file video finiva prima del dovuto ed altri.

```
sock.sendto(base64.b64encode(pack), address)
```

Cosi ho pensato di usare la liberira *Base64* per spedire e ricevere i pacchetti effettuando le corrette conversioni con questo modo sono riuscito ad inviare e scrivere correttamente i pacchetti, l'ultima cosa da correggere era la chiusura del file, ovvero far capire al client che i pacchetti erano finiti e che poteva continuare con i suoi altri compiti, cosi ho pensato di inviare un pacchetto di terminazioni alla fine dei pacchetti del file selezionato, pacchetto di terminazione che il client avrebbe riconosciuto e cosi si sarebbe fermato.

Una volta riuscito ad inviare e scrivere correttamente i pacchetti il resto è stato più semplice, ho trasformato in funzioni riutilizzabili il codice ed ho scritto un piccolo main per la ricezione della scelta del comando effettuata dall'utente, ed in base alla diversa scelta chiama le dovute funzioni

Funzioni e comportamento del server

Il server ha quattro funzioni principali ed un main:

- Connect_server
- File_list_server
- Get_server
- Put_server

Main

```
sock = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
address = ('localhost', 10000)
sock.bind(address)
try:
    Connect_server()
    while True:

        choose_com, address = sock.recvfrom(4096)
        if int (choose_com.decode())>=1 and int(choose_com.decode())<=4:
            sock.sendto(("OK").encode(), address)
            if int(choose_com.decode()) == 1:
                print ("è stato scelto il comando '1 - visualizzare i file presenti sul server'\n" )
                files=File_list_server()
            elif int(choose_com.decode()) == 2:
                print("è stato scelto il comando '2 - Scaricare un file presente sul server'\n")
                print (Get_server())

            elif int(choose_com.decode()) == 3:
                print ("è stato scelto il comando '3 - Caricare un file sul server'\n")
                print(Put_server())

            elif int(choose_com.decode()) == 4:
                print ("è stato scelto il comando '4 - Chiusura del server'\n ")
                break

        else:
            sock.sendto(("ER").encode(), address)

except Exception as error:
    print (error)
finally:
    sock.close()
```

Il main benchè sia il blocco più lungo è il piu semplice inzialmente viene inizializzato il socket, dopodichè chiama la Connect_server che se da esito postivo si connette al client e fa continuare il main, attende di sapere quale comando è stato scelto e appena gli arriva un messaggio controlla che il comando scelto sia presente tra quelli proposti, in caso contrario invia al client un messaggio di errore e attende un nuovo comando, se il comando è valido il main lo identifica e chiama la funzione corrispondente.

Connect_server

```
def Connect_server():
    while True:
        print ("Server in ascolto in attesa della connessione del client ")
        start, address = sock.recvfrom(4096)
        if start :
            sock.sendto(("OK").encode(), address)
            print("Server connesso" )
            return (None)
        else:
            sock.sendto(("ER").encode,address)
            print("Connessione con il client fallita, riprovare"
```

La funzione Connect_sever viene chiamata dal main all'avvio dell'applicazione, rimane in ascolto finchè un client non si collega, se avviene o no il collegamento manda un messaggio con l'esito positivo o negativo che sia, in caso di collegamento riuscito la funzione termina e il main continua le sue operazioni, nel caso in cui non si colleghi rimane in ascolto per un altro eventuale client.

File_list_server

```
def File_list_server():
    print ("Invio la lista dei file presenti sul server \n")
    files = os.listdir()
    for sendable_file in files:
        if(sendable_file == "Server.py"):
            continue
        sock.sendto(sendable_file.encode(), address)

    files.remove("Server.py")
    sock.sendto("END".encode(), address)
    print ("Iinvio riuscito")
    return (files)
```

La funzione crea una lista contenente tutti i file presenti nella cartella del server, dopodichè invia i nomi dei file al client saltando il server stesso, una volta finito l'invio viene rimosso il server dalla lista in modo che possa richiamare questa funzione anche per avere una lista dei file e che la possa usare per l'invio dei file.

Get_server

```
def Get_server():
    files=(File_list_server())
    while True:
        chosen_file, address =sock.recvfrom(4096)
        if files[int(chosen_file.decode()) - 1] in files:
            sock.sendto(("OK").encode(), address)
            print("E' stato scelto il file", chosen_file.decode(),
                          files[int(chosen_file.decode()) - 1],"\n")
            file = open(files[int(chosen_file.decode()) - 1], 'rb')
            print ("Invio il file selezionato \n")
            pack = file.read(1024)
```

```
            while pack:
                sock.sendto(base64.b64encode(pack), address)
                pack=file.read(1024)
                time.sleep(0.0005)
            sock.sendto(base64.b64encode("END".encode()), address)
            file.close()
            return ("File inviato correttamente \n")
        else:
            print ("Il file scelto non è presenta sul server sceglierne un altro \n")
            sock.sendto(("ER").encode(), address)
```

Questa funzione viene chiamata quando un file deve essere scaricato dal server, come prima cosa richiama File_list_server per avere una lista con tutti i file, dopodichè attende di sapere dal client quale file è stato scelto e controlla che il file sia presente sul server in caso negativo invia un messaggio di errore e attende una nuova scelta, se invece il file scelto è presente sul server lo apre il lettura inizia a leggerlo e invia con le dovute conversioni i pacchetti al client, una volta che ha finito di inviare i pacchetti del file ne invia un ultimo con il messaggio “END”, ovvero il messaggio di terminazione.

Put_server

```
def Put_server():
    file_recv, address =sock.recvfrom(4096)
    file = open(file_recv.decode(), 'wb')
    pack, server =sock.recvfrom(4096)
    while(base64.b64decode(pack.decode())!= "END".encode()):
        if(base64.b64decode(pack.decode()) != "END".encode()):
            file.write(base64.b64decode(pack.decode()))
        pack, address = sock.recvfrom(4096)
    file.close()
    return("File scaricato correttamente")
```

Analogamente a quando il client riceve un messaggio Put_server fa lo stesso, prima attende di sapere il nome del file che riceverà in modo da aprire un file con quel nome in scrittura, dopodichè inzia a ricevere i pacchetti finchè uno di questi non conterrà il messaggio di terminazione, ad ogni lettura del pacchetto che non fa finire il ciclo viene scritto nel file precedentemente aperto, una volta finita la ricezione il file viene chiuso e viene inviato un messaggio di esito positivo.

Funzioni e comportamento del client

Il client è composto da 4 funzioni ed un main :

- Connect_client
- File_list_client
- Get_client
- Put_client

Main

```
sock = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
address = ('localhost', 10000)
try:
    print(Connect_client(address))
    list_mod=("\nComandi disponibili:\n "
              "1 - Visualizzare i file presenti sul server\n "
              "2 - Scaricare un file presente sul server\n "
              "3 - Caricare un file sul server\n "
              "4 - Chiusura del server")
    while True:
        print(list_mod)
        choose_com = int(input("Inserisci il numero del comando che vuoi eseguire: "))
        sock.sendto(str(choose_com).encode(), address)
        control, address = sock.recvfrom(4096)
        if control.decode() == "OK":
            if choose_com == 1:
                List_files_client()
            elif choose_com == 2:
                print(Get_client(address))
            elif choose_com == 3:
                print(Put_client())
            elif choose_com == 4:
                break
        else:
            print ("Scegliere un comando valido \n")
except Exception as info:
    print(info)
finally:
    print("Chiudo il client")
    sock.close()
```

Come prima cosa viene chiamata la funzione Connect_client che se va a buon fine connette il client al servere fa ritorna un esito positivo, similarmente al main del server quello del client compie le stesse operazioni, la differenza è che nel client viene chiesto quale comando l’utente voglia eseguire, la scelta viene poi inviata al server e se la scelta è corretta allora il client riceverà un messaggio di esito postivo che fara continuare il programma e dopo aver identificato il comando verrà chiamata la funzione che ne corrisponde, in caso in cui il messaggio di ritorno dal server per la scelta del comando sia negativo, viene fatto notare all’utente e si ripete la scelta.

Connect_client

```
def Connect_client(address):
    print ("Client pronto, mi connetto al server \n")
    sock.sendto(("OK").encode(), address)
    control, address = sock.recvfrom(4096)
    if control.decode() == "OK":
        return ("Client connesso al server")
    else:
        print("Connessione fallita riprovare")
```

Questa funzione invia un messaggio al server per connettersi, aspetta un esito di ritorno dal server, se postivo ritorna che la connessione è riuscita, altrimenti stampa un messaggio di errore.

List_files_client

```
def List_files_client():
    print("\nLista dei file disponibili:")
    list_file = []
    num = 1
    files, server = sock.recvfrom(4096)
    while(files.decode() != "END"):
        print(num, files.decode())
        list_file.append(files.decode())
        files, server = sock.recvfrom(4096)
        num+=1
    return (list_file)
```

Questa funzione viene chiamata quando si vogliono sapere i file presenti sul server, come prima cosa crea una lista dove salva tutti i file per essere reperibili poi da altre funzioni, dopodichè aspetta di ricevere file per file finchè non arriva un pacchetto con il messaggio di terminazione, per ogni pacchetto riccevuto viene stampato a schermo e salvato nella lista, alla fine della funzione si ritorna la lista completa.

Get_client

```
def Get_client(address):
    files=List_files_client()
    while True:
        choose_file = int(input("Inserisci il numero del file che vuoi ricevere: "))
        if choose_file>=1 and choose_file<=len(files):
            file = open(files[choose_file - 1], 'wb')
            sock.sendto(str(choose_file).encode(), address)
            control, address = sock.recvfrom(4096)
            if control.decode() == "OK":
                print ("\nScaricando il file "+ files[choose_file-1])
                pack, server =sock.recvfrom(4096)
                while(base64.b64decode(pack.decode())!= "END".encode()):
                    if(base64.b64decode(pack.decode()) != "END".encode()):
                        file.write(base64.b64decode(pack.decode()))
                        pack, server = sock.recvfrom(4096)
                file.close()
```

```
                return("\nIl file "+ files[choose_file-1]+ ù
                            " è stato scaricato correttamente")
        else:
            print("\nIl numero scelto non corrisponde a nessun file presente sul server",
                                        "selezionarne un'altro")
```

Il compito di questa funzione è di scaricare e salvare un file presente sul server, inizialmente crea una lista chiamando la funzione List_files_client che ritorna tutti i file presenti sul server, poi dopo aver appreso la scelta dell'utente su quale file scaricare lo apre in scrittura ed invia al server la scelta dell'utente in modo che anche lui possa aprire lo stesso file in lettura ed inziare ad inviare i pacchetti, infatti dopo aver avuto conferma che il file scelto sia nel server inizia a ricevere pacchetti e scriverli, questo fino all'arrivo del carattere di terminazione, dopo cio chiude il file e ritorna un messaggio con l'esito positivo dell'operazione.

Put_client

```
def Put_client():
    files= os.listdir()
    files.remove("Client.py")
    while True:
        print("\nLista dei file disponibli:")
        num=1
        for i in files:
            print (num,i)
            num+=1
        choose_file=int(input("Inserisci il numero del file da inviare: "))
        if choose_file>=1 and choose_file<=len(files):
            sock.sendto(files[choose_file - 1].encode(), address)
            print("E' stato scelto il file", choose_file, files[choose_file - 1],"\n")
            file = open(files[choose_file - 1], 'rb')
            print ("Invio il file selezionato \n")
            pack = file.read(1024)
            while pack:
                sock.sendto(base64.b64encode(pack), address)
                pack=file.read(1024)
                time.sleep(0.0005)
            sock.sendto(base64.b64encode("END".encode()), address)
            file.close()
            return ("Il file " + files[choose_file-1] +
                      " è stato inviato correttamente \n")
        else:
            print("Il numero scelto non corrisponde a nessun"
                 "file presente nel client, selezionarne un'altro")
```

Questa funzione svolge lo stesso compito di Get_server ovvero, prima di tutto crea una lista con i file presenti sul client e la stampa per far scegliere all'utente cosa inviare al server, una volta scelto il file viene mandato un messaggio al server con il nome del file e viene aperto dal client in lettura, poi inizia ad essere letto ed inviato, alla fine dei paccchetto viene mandato un messaggio di terminazione.