RELAZIONE

Il progetto consiste nella creazione di una chat multiutente che sia tollerante agli errori. Si dovranno sviluppare due applicazioni, lato client e lato server. Quest’ultima si occuperà della ricezione dei messaggi, l’astrazione dei servizi forniti all’utente e della sincronizzazione con un altro server, il quale sarà obbligatorio al fine del funzionamento del servizio, poiché fondamentale nel bilanciamento del carico e nella gestione dei vari errori di connessione lato server. Il lato client invece si occuperà dell’invio dei messaggi, la decisione di eventuali spostamenti da un server al altro per problemi dettati dalla congestione o dal non funzionamento di uno dei due server e di fornire una grafica all’utente.



Quindi ho scelto di implementare questo progetto con un’architettura Client-Server. Dove il server ha ruolo di mainframe per la gestione di tutta la messaggistica, infatti i dati sono salvati solo lato server a differenza del lato client che possiede solo una cache, che perde ogni qualvolta che si disconnette, ma riceve tutti i messaggi una volta connesso al server. Ho scelto quest’architettura per snellire il client che oggettivamente sarebbe risultato più complesso se sviluppato per un’architettura P2P, scegliendo questo tipo d’architettura si sarebbe andato incontro anche ad una duplicazione d’informazioni ed a una maggiore complicazione per il recupero dei messaggi quando non si è collegati nella chat e non si sarebbe potuta garantire la fault tolerance. Per inviare questi pacchetti ho scelto il TCP, poiché dev’essere garantito l’arrivo del messaggio al server e dal server al client, non solo per questo, ma anche perché solamente il TCP mi garantisce un metodo di connessione FULL Duplex, quindi che riesco ad inviare e ricevere tramite due canali separati, ciò mi permette di ricevere messaggi in real-time. Per implementare tutto ciò ho usato Python 3.8 e le sue librerie di sistema (socket, threading e tkinker).

Il lato server è scomposto in due socket, interfacce che ci permettono di comunicare con il livello superiore dello stack protocollare, il primo socket si occupa dell’invio e ricezione da parte dei client, il secondo della sincronizzazione con il server di backup. Il server prima di poter essere operativo attende che il server di backup sia sincronizzato con lui, tramite una connessione tipo master/slave, dove entrambi saranno sia master che slave, master chi invia e slave chi riceve. Una volta sincronizzati i due server iniziano ad accettare un massimo di 100 utenti, i quali vengono aggiunti alla lista dei client connessi e si istanzierà un thread per gestire la comunicazione con esso. Nella gestione della comunicazione con esso il server registra il suo nome ed invia tutto il suo archivio, con un ciclo infinito gestisce la ricezione e l’invio, quest’ultimo verrà fatto in un simil broadcast, poiché andiamo ad escludere il mittente. Lato sincronizzazione il thread preposto riceve il messaggio dall’altro server e lo invia in simil broadcast a tutti i client, anche qui alla prima connessione viene inviato tutto l’archivio.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Da riga 174-176 abbiamo il loop infinito che ci permette che entrambi i server siano connessi l’uno con l’altro, infatti terminerà quando la funzione sync ritornerà True. Il ritorno della funzione avverrà quando il client di backup (il secondo server) sarà disponibile per la connessione, altrimenti il ciclo continuerà ad eseguirsi con intervalli di 2 secondi.

Immagine che contiene testo

Descrizione generata automaticamente

La funzione syncro viene utilizzata al fine da avere una sincronizzazione in real-time tra i due server, essa effettua un loop infinito dove al suo interno se riuscisse a ricevere un messaggio proveniente dalla socket del server di backup (client è la socket del server di backup), effettuerà la decodifica in utf-8 (codifica di caratteri), appenderà il messaggio decodificato all’archivio ed invierà il messaggio alla funzione send

Immagine che contiene testo

Descrizione generata automaticamente

Questa funzione effettua una copia della lista dei client connessi al server, inizia un ciclo for che controlla ogni elemento della lista, a riga 55 viene effettuato un controllo affinché non si rimandi al mittente il messaggio, in modo da scongiurare un rimbalzo infinito di questo messaggio. Dopo di che esso viene spedito al client, nel caso dovesse riscontrarsi un qualsiasi problema il server chiuderà la connessione con il client in questione e lo rimuoverà dalla sua lista.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Questa funzione, nettamente la più complessa, è la gestione di tutta la ricezione dei messaggi ed anche l’invio. La funzione inizia con un controllo sul numero dei client connessi a riga 70, esso ci permette di sapere quanti client ci sono connessi e comunicarlo al client, affinché sia lui a decidere in quale connettersi. A riga 71 c’è un controllo affinché non venga inviato al server di backup, l’assenza di questo controllo causerebbe un invio di un carattere non codificato. Una volta inviato il nostro stato il client ci invierà il suo Nick, che verrà usato al fine di identificarsi con gli altri utenti. A riga 77 si nota come si controlla se il client è o meno quello di backup al fine di identificare il client a cui mandare tutti i messaggi dell’archivio, poiché il server di backup li prenderà dal file sulla memoria di massa. A riga 94 inizia il ciclo che ci permette la gestione di tutti i messaggi, il server riceve il messaggio ed effettua un controllo (a riga 98) sulla sua lunghezza, questo ci permette di scovare errori sulla trasmissione in ricezione. Se non si presenteranno errori a riga 100 verrà effettuato un controllo al fine di capire se il client vuole effettuare una chiusura della connessione, se ciò si verifica viene confermata la chiusura lato server e viene rimosso dalla lista dei client. Altrimenti il messaggio verrà formattato in uno standard comune, aggiunto all’archivio, salvato nel file sulla memoria di massa ed inviato in broadcast tramite la suddetta funzione, che ha come argomento il messaggio e la socket.

Immagine che contiene testo

Descrizione generata automaticamente

Essa effettua un processo analogo alla funzione Send()

Immagine che contiene testo

Descrizione generata automaticamenteIl lato client è scomposto anche questo in due socket identici al lato server. Qui abbiamo anche la grafica che è stata sviluppata tramite l’utilizzo di una classe per semplificare il suo sviluppo. Il client si connetterà al server tramite gli argomenti dati da riga di comando, dopo aver creato il socket con questi dati ed effettuato l’handshake a tre vie inizia la trattativa con il server per il bilanciamento. Il server invia il numero di client connessi ad esso, che sarà pari ad N+1 (dove 1 è il server di riserva), il client deciderà su un determinato target di cambiare il server nel successivo. Una volta stabilita la connessione viene inizializzata tutta la grafica e viene chiesto all’utente di inserire il nome con la quale vuole partecipare alla chat, dopo di che il thread di ricezione viene avviato e rimarrà in un loop infinito nell’attesa di un messaggio, ma per garantire al fault tollerance ci sarà un time-out (di 200 secondi, andrebbe stimato meglio) che ci determina se il server è ancora in rete e non ha nessun problema, esso ha pure il dovere di stampare nella textBox i messaggi ricevuti.

Questa funzione gestisce tutta l’inizializzazione del socket, compresa la gestione degli eventuali errori. A riga 38 vediamo come tramite un flag si decide come modificare la porta questo ci serve per quando viene richiamata durante la gestione dell’eccezione nell’invio o nella ricezione, il primo caso vuol dire che posso impostare come è stato scelto dall’utente in fase d’avvio, il secondo farò uno switch tra le due porte (attualmente si considera siano conosciute). A riga 44 vediamo come si presenta un ulteriore controllo, questo ci dice se i server alla quale ci stiamo collegando si trovino su di un’altra macchina, in caso essa si verifichi il nuovo indirizzo verrà richiesto all’utente.

A riga 48 effettuiamo la connessione, con la socket precedentemente impostata, subito dopo riceviamo il numero di utenti collegati al fine di bilanciare il carico, a riga 51 si effettua il controllo per un numero limite imposto da noi arbitrariamente, se esso si verifica chiudiamo la nostra socket per crearne un’altra tramite il parametro passato per riga di comando. Si presentano 3 casi:

* Il primo quando l’argomento è uguale a 0, quindi i server si trovano sulla stessa macchina e con lo stesso indirizzo.
* Il secondo quando l’argomento è uguale ad 1, quindi i server si trovano su due macchine diverse con due indirizzi differenti e porte anch’esse differenti.
* Il terzo quando i server si trovano su due macchine diverse con indirizzi differenti, ma porte uguali.

Una volta formati gli argomenti della socket, essa viene ricreata ed usata per la connessione al server. Il quale ripeterà nuovamente tutti i controlli.

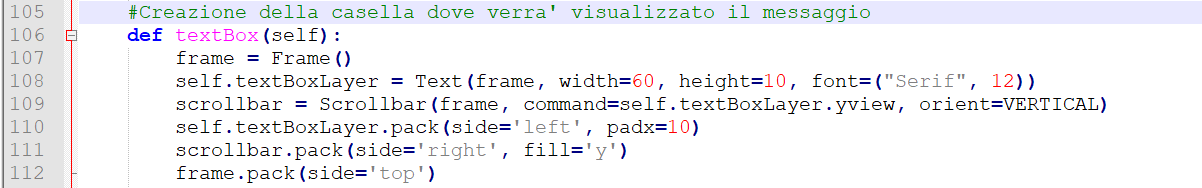
Immagine che contiene testo

Descrizione generata automaticamente

Queste due funzioni servono al fine di modificare gli argomenti della nostra socket.

Immagine che contiene testo

Descrizione generata automaticamente

La funzione gui\_Initializer(), serve al fine di inizializzare tutta la parte grafica del progetto. Infatti, richiama tutti gli elementi che vediamo nella nostra schermata. A riga 67 vediamo il nome della finestra, successivamente il richiamo alla funzione atta alla creazione della chatBox, la casella dove l’utente scriverà il suo testo da inviare.

A riga 107 viene creato il nostro frame, quindi lo spazio dove si andrà a leggere i messaggi ricevuti, a riga 109 la scrollbar che ci permette di scorrere i messaggi. A riga 110 vengono date le impostazioni di posizionamento e visualizzazione, successivamente della scrollbar e di tutto il frame.

La stessa procedura viene effettuata per le altre funzioni, ma si deve notare un particolare nella chatbox che viene ripreso anche nella username\_Section.



Come vediamo a riga 120 viene creato un bottone nel frame di riferimento che avrà come comando l’invocazione della funzione on\_TextSend(). Questa funzione risveglia il thread d’invio, dopo di che pulisce la chatbox, precisamente a riga 132 e 133.Immagine che contiene testo

Descrizione generata automaticamente

La funzione che viene richiamata a riga 132, quindi la send\_MSG, è atta a tutta la gestione dell’invio dei vari messaggi da parte dell’utente e della formattazione per la visualizzazione in chat del proprio messaggio inviato. A riga 140 la funzione prende l’username, già precedentemente scritto, e lo formatta con la stringa data, la quale verrà acquisita dalla chatbox. A riga 142 notiamo un controllo sulla chiusura della connessione, al fine negoziare una chiusura con il server, anche se il server quando riscontrerà un problema nella comunicazione con noi ci escluderà dalla sua lista automaticamente. A riga 146 il messaggio viene codificato in byte al fine di essere trasmesso, nelle due righe successive viene inserito il messaggio correttamente formattato nella nostra textBox. A riga 149 avviene l’invio del messaggio al server, se in questo invio dovesse verificarsi un errore il sistema chiuderà la connessione con il server e richiamerà la funzione per inizializzare il socket passando come argomento dello status 0, quindi il dover cambiare porta ed eventualmente indirizzo.

Il thread d’invio verrà risvegliato ogni volta che si preme il bottone invio, in modo da gestire tutto l’encoding e la formattazione del messaggio.Immagine che contiene testo

Descrizione generata automaticamente

Qui notiamo come a riga 79 viene impostato il timeout, quindi un timer che conta per 200 secondi come detto precedentemente per il corretto controllo di possibili problemi lato server. Se esso dovesse scadere, quindi passano i 200secondi, verrà prontamente chiusa la connessione e richiamata l’inizializzazione della socket.