Documentazione Progetto Reti Informatiche

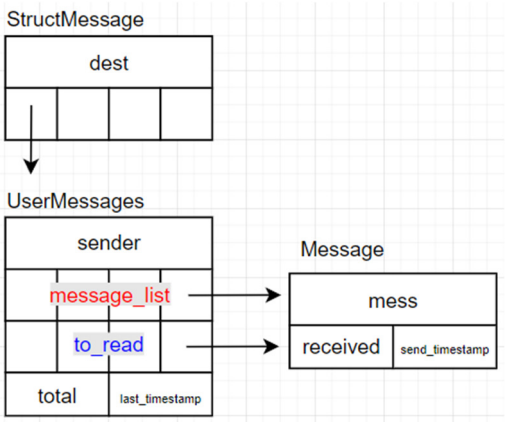
Dario Zippo(589258)

Strutture Dati:

Una struttura dati utilizzata in comune sia dal Server che dai client è **VECTOR**, che rimarca campi e metodi presenti nella sua corrispettiva classe definita nelle librerie standard C++. Il ruolo è stato sostitutivo alle liste.

Server: Per memorizzare la lista degli utenti utilizzo il vettore userRegister, che memorizza oggetti di tipo **Record**. Tale struttura contiene gli elementi definiti nelle specifiche, ovvero username, porta, timestamp login, timestamp logout, ed in aggiunta il socket associato a quell'utente, per poter aggiornare il suo timestamp logout in caso di disconnessione del client, ed al tempo stesso conoscere il socket a cui inviare una notifica di show nel caso l'utente sia online. Le disconnessioni improvvise sono gestite controllando dopo ogni recv se il valore ritornato è 0.

Per i messaggi pendenti utilizzo un vettore di **StructMessage**, una per ogni destinatario. Ogni elemento presenta un vettore di UserMessages, una per ogni mittente, contenente la lista dei messaggi inviati e quella di quelli ancora da leggere, salvati tramite Vector. Ogni messaggio è un oggetto caratterizzato dalla stringa inviata, un booleano di ricezione ed il relativo timestamp



Vi è poi poi il vettore UserLinks, contenente elementi **UserLink**, caratterizzati da mittente e destinatario. Il vettore serve per la gestione delle notifiche postume a l'uso di uno show da parte di un utente con messaggi pendenti che lo designano come destinatario.

Device: Per descrivere dello stato attuale del client come coinvolgimento in chat P2P individuali o di gruppo, e per la memorizzazione delle informazioni necessarie alla gestione di tali chat, utilizzo la struttura dati **ChatState**.

File utilizzati:

Server:

- **user.txt** contiene tutti gli utenti registrati
- **login.txt** file di log contenente tutti i login/logout effettuati
- **saved_messages.txt** contiene tutti i messaggi pendenti, viene scritto quando il server fa esc e letto all'avvio del server per inizializzare le strutture dati dedicate ai messaggi

Device:

- Per ogni user vi è un file di rubrica **contacts/username.txt**
- Per ogni user vi è un file di storico dei messaggi inviati per ogni utente destinatario, chiamato **chat_username/dest_username.txt**

Protocolli e casi d'uso:

L'applicazione utilizza solo socket TCP per garantire una comunicazione affidabile dei messaggi.

Gli username e i messaggi hanno una dimensione massima di 1024 caratteri.

Ogni client subito dopo aver stabilito la connessione e aver fatto il login, apre un socket in ascolto sulla porta num_porta in modo che possa accettare richieste di connessione di altri client.

Per ogni operazione vi è un intero relativo. L'utente che vorrà usare una delle funzionalità del device/server, inserirà l'intero relativo nel terminale, successivamente avrà le istruzioni su come continuare. Anche il device comunica comandi al server tramite interi relativi a specifiche funzioni. Qui le rispettive liste di cui l'utente deve essere a conoscenza.

Device:

1. **signup**
2. **in**
3. **hanging**
4. **show**
5. **chat**
6. **out**

Server:

1. **help**
2. **list**
3. **esc**

Fase di accesso: All'avvio il client, dopo aver stabilito una connessione con server si bloccherà su di un menù di accesso/registrazione, da cui uscirà solo dopo aver effettuato il login, salvandosi il proprio username corrente. Per richiedere l'operazione al server dovrà prima digitare numero relativo (1/2), che aprirà il form di accesso/registrazione, dove dovrà inserire "username password". Verrà poi inviato il tutto al server. In caso di login il server risponderà con un feedback: "LOGIN"(positivo), "NO"(negativo).

Hanging: Il server, ricevuto il comando e lo username dell'utente che l'ha mandato, scorrerà il vettore UserMessages, ovvero la lista dei mittenti, per trarne le info ed inviare stringhe di 1055 byte (username(1024) num.mess.(5) timestamp(24)), presentate in questo formato: "**Username Numero Messaggi inviati Timestamp piu' recente".** Il server comunicherà quando non troverà mittenti di messaggi pendenti inviando una stringa "ZERO"

Show: Il server, ricevuto il comando e gli username, invia al device prima il numero di messaggi pendenti da leggere, e poi li invia, aggiornando poi le strutture dati.

Chat:

Quando un client esegue il comando **chat username**, il server risponde con una porta.

- Username è offline: la porta = 0 ed il client manderà messaggi pendenti al server
- Username è online: riceve dal server la porta su cui contattare username, quindi inizia chat Peer-to-Peer

Per far avvenire la chat Peer-to-Peer si eseguono i seguenti passi:

- Ogni peer dopo aver eseguito login crea socket di ascolto (IP: localhost, porta passata come argomento)
- Peer mittente crea nuovo socket adibito alla comunicazione con il peer destinatario
- Peer mittente esegue connect con peer destinatario (IP: localhost, porta l'ha ricevuta precedentemente dal server)
- Peer destinatario con accept crea il socket di comunicazionione

Durante una chat P2P, un device può inviare specifici messaggi preceduti da carattere d'escape:

- **q** Comando di chiusura della chat.
- **u** Richiesta al server per ottenere la lista degli utenti online, per sapere chi posso scegliere per l'aggiunta ad una group chat
- **a username** se username è online si può creare il gruppo che conterrà inizialmente i due peer che stavano chattando e username. Durante una chat di gruppo un nuovo componente può essere aggiunto sempre digitando \a username, il gruppo può contenere fino a 10 componenti, si assume che gli utenti rispettino questo vincolo.

Si può partecipare a una sola chat di gruppo per volta, i gruppi non hanno un nome, quando uno dei componenti del gruppo esce dal gruppo digitando \q il gruppo smette di esistere. Le chat di gruppo hanno la priorità sulle chat normali, se un utente stesse chattando con un altro, e venisse aggiunto a un gruppo, i messaggi che invio da quel momento in poi sono indirizzati al gruppo.

Quando si riceve o si digita \q si esce non solo dal gruppo ma anche delle eventuali chat attive. Se ci sono più chat attive (e nessun gruppo) i messaggi che digito sono diretti all'ultimo peer con cui ho avuto uno scambio di messaggi (questa cosa vale sempre eccetto che per il primo messaggio, inviato allo user specificato nel comando)

ESC: Il server aggiorna i logout timestamp con quello corrente. Dopo il "riavvio" non ci saranno quindi utenti connessi ad esso. Tale scelta è motivata dal fatto che considerare online gli utenti che erano connessi al momento della ESC vorrebbe dire ignorare la possibilità che parte di essi, se non tutti, possano essersi disconnessi nell'intervallo di tempo tra lo spegnimento ed il riavvio del server. Ciò potrebbe comportare gravi conseguenze di inconsistenza, in quanto potrei portare dei device ad avviare chat P2P con utenti offline. Questa scelta comporta comunque altri tipi di inconsistenza,

come ad esempio salvare messaggi pendenti per utenti che in realtà sono online, in quanto non disconnessi durante la ESC, ma ho trovato il compromesso accettabile.

Se il server è in ascolto su una porta diversa dalla 4242 si suppone che i client lo sappiano e lo passino come secondo argomento quando eseguono l'applicazione ./device mia_port serv_port