DOCUMENTAZIONE PROGETTO RETI INFORMATICHE a.a. 2021-2022

NICCOLÒ AMATO – 596872

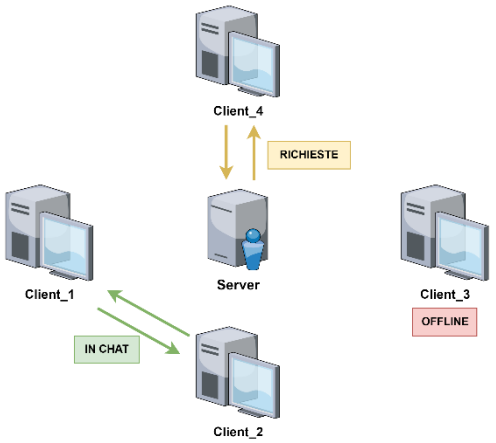

Figura 2: scenario applicativo

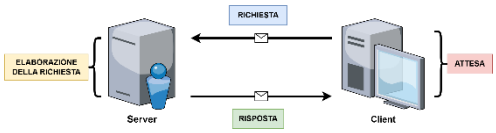

Figura 1: esempio di scambio di messaggi

1. Descrizione generale

Il progetto consiste nello sviluppo di un'applicazione di instant messaging basata su un paradigma ibrido peer-to-peer e client-server. L'applicazione implementa le funzionalità di notifiche, messaggistica online e offline, chat di gruppo e sharing di file testuali. La comunicazione avviene tramite scambio di messaggi attraverso un protocollo di invio della richiesta, elaborazione della richiesta e invio del messaggio di risposta mostrato schematicamente in *Figura 2*, mentre in *Figura 1* è rappresentato un tipico scenario applicativo.

2. Sviluppo

Il server mantiene in memoria una struttura dati utile per memorizzare *username*, *porta*, *socket di connessione*, *timestamp di login* e *timestamp di logout* degli utenti che fanno il login. In modo analogo ogni device ha in memoria una struttura dati dove verranno memorizzati *username*, *porta*, *socket di connessione* degli utenti con cui viene iniziata una chat. Per lo sviluppo dell'applicazione ho usato dei **socket TCP bloccanti**. Infatti, essendo l'applicazione basata sullo scambio di messaggi, ho voluto dare priorità alla consistenza di questi. La scelta di fare socket bloccanti è dovuta alla logica del protocollo che sta alla base del progetto che prevede che quando viene inviato un messaggio (una richiesta) ci si aspetta sempre una risposta (dal device o dal server). Pertanto risulta fondamentale l'utilizzo della ***select*** nel *main* sia del server sia dei device. Il server è mono-processo, questa scelta progettuale porta con se i problemi del singolo *point of failure* e le *attese* da parte dei device. Una soluzione alternativa avrebbe potuto essere lo sviluppo del progetto utilizzando i *thread*. Per la gestione della chat, ciascun device, ha un array di interi che viene usato per salvare i socket di connessione con gli altri utenti così da poter inviare messaggi a tutti gli utenti in chat (nel caso di chat di gruppo).

3. Signup

Ciascun utente può iscriversi nell'applicazione attraverso il comando di signup. Il formato richiesto è il seguente: *signup username password*. Non possono esistere più utenti con lo stesso username. Se la richiesta è inviata correttamente, il server aggiornerà il file *users.txt* dove andrà a memorizzare l'username del nuovo utente iscritto e il file *utenti_iscritti.txt* dove verranno memorizzati *username* e *password* del nuovo utente. Solo gli utenti iscritti possono effettuare il login.

4. Login

Il server riceve la richiesta di login e, se la richiesta è stata inviata in modo corretto, invia un *ack* di login avvenuto con successo al client. Oltre a ciò, invia una *notifica* nel caso in cui il client avesse iniziato una chat con un client offline che successivamente ha letto i messaggi ricevuti, controllando se è presente nella cartella dell'utente un file *messaggi_inviati.txt*. Se è la prima volta che l'utente effettua il login (da quando il server è attivo) il server salverà nella prima entry libera della struttura dati degli utenti le informazioni dell'utente che si è loggato.

Il formato del messaggio è il seguente:

in	4242	username	password	mio_username	porta_device

5. Hanging

Un utente online può richiedere di vedere se ha ricevuto messaggi mentre era offline attraverso il comando hanging. Il server gli invierà in risposta un ack di hanging avvenuto con successo insieme a un buffer contenente una lista degli utenti che hanno messaggi pendenti.

Il formato del messaggio è il seguente:

hanging	mio_username

5.1. Show

Dopo aver fatto hanging un utente può richiedere di visualizzare i messaggi ricevuti da uno specifico utente mentre era offline inviando al server la richiesta di show.

Il formato del messaggio è il seguente:

show	nome_file	username

5.2. Chat

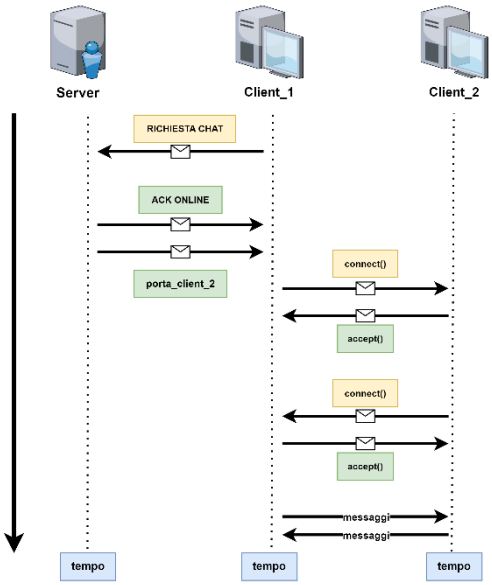

Figura 3: chat online

Un utente online può aprire una chat con un altro utente solo se lo ha salvato in rubrica. In quel caso il server invia al richiedente un ack di utente online o utente offline. Se l'utente con cui si vuole parlare è online, il client mittente riceve dal server la porta del client con cui vuole chattare ed esegue una connessione con quel client e salva nella struttura dati degli utenti i dati dell'utente con cui desidera chattare. L'altro client riceverà dei messaggi di notifica e se volesse entrare in chat con il primo utente dovrebbe effettuare anche lui la richiesta di chat per potersi connettere. Se entrambi effettuano la richiesta di chat allora potranno chattare tra di loro senza la mediazione del server. Se l'utente con cui si vuole parlare è offline, si aprirà una chat con il server che salverà i messaggi in un file di messaggi pendenti.

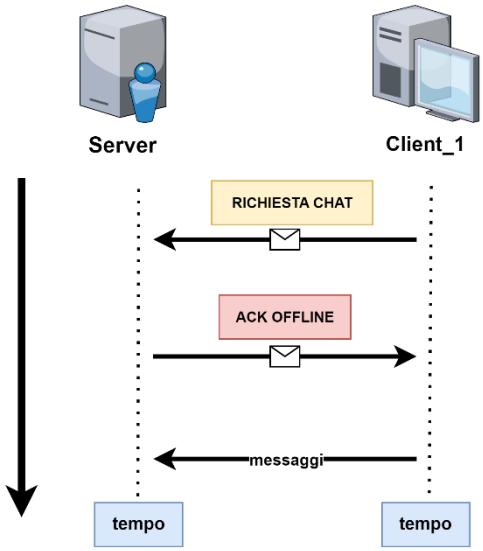

Figura 4: chat offline

In chat, gli utenti possono vedere quali altri utenti sono online attraverso il comando */u* e nel caso creare delle chat di gruppo aggiungendo alla chat utenti online attraverso il comando */a username*. In questo caso l'utente che viene aggiunto alla chat accetta le connessioni degli utenti già presenti in chat e a sua volta richiede la connessione con loro. Come detto precedentemente, i socket di connessione sono gestiti con un array di interi che per ogni messaggio viene controllato e usato per ottenere i socket di connessione corretti.

Il formato del messaggio è il seguente:

chat	username

5.3. Share

Gli utenti in chat possono condividere dei file di testo in loro possesso (presenti nella loro cartella). La funzione di share consiste nel copiare il contenuto del file che si vuole condividere e ricopiarlo in un file omonimo che viene creato nella cartella degli utenti destinatari. Se un utente condivide un file in una chat di gruppo, questo file sarà condiviso anche con gli utenti presenti nella chat di gruppo anche se non sono salvati nella rubrica del mittente. Esempio: per condividere il file “prova.txt” bisogna scrivere *share prova*.

5.4. Esc

Il server può disconnettersi attraverso il comando Esc. In tal caso il server toglie tutti i socket di comunicazione dalla lista *master*, chiude il proprio socket e termina il processo. Quando il server si disconnette, gli utenti in chat possono continuare a scambiarsi messaggi tra di loro in quanto le chat avviate sono indipendenti dalle funzionalità del server. In ogni caso non sarà possibile fare il login, una nuova iscrizione, iniziare una nuova chat, né richiedere l’hanging o lo show dei messaggi pendenti.

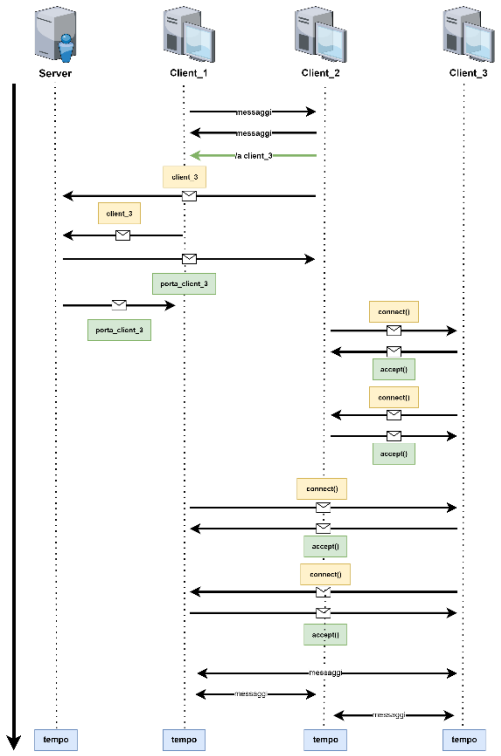

Figura 5: chat di gruppo

6. Gestione delle disconnessioni improvvise

Le disconnessioni improvvise dei dispositivi chiudono e rendono sempre pronto il loro socket. Pertanto, la select troverebbe il socket del dispositivo che si è disconnesso sempre pronto nel master. Ho implementato quindi una funzione isSocketValid che restituisce un valore booleano. Se il socket da cui si riceve il messaggio è il socket del dispositivo che si è disconnesso la funzione restituirà il valore 0. In tal modo si può capire se il socket è di un dispositivo che si è disconnesso in modo anomalo o meno, e nel caso lo si toglie dal master, così che la select possa funzionare correttamente.