UNIVERSITÀ DI PISA

RELAZIONE PROGETTO RETI INFORMATICHE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
A.A. 2021/2022

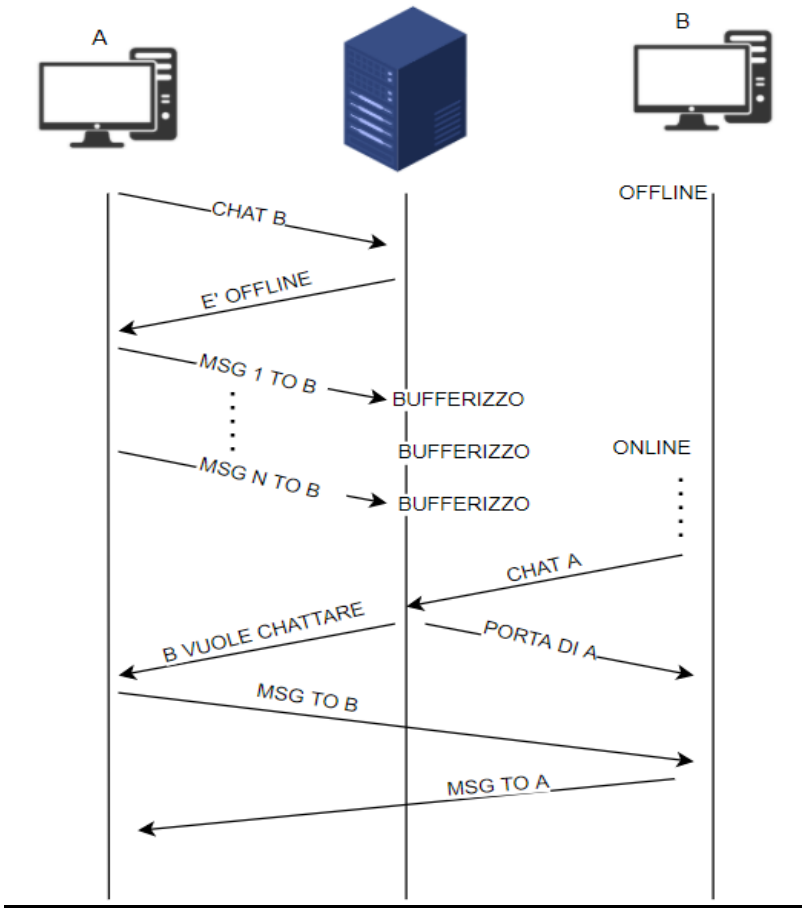
Studente, matricola:

Marcuccetti Gabriele, 603303

Il progetto riguarda un'applicazione di instant messaging, in cui sia client che server utilizzano I/O multiplexing per gestire le richieste dei client e l'input dell'utente, utilizzando socket bloccanti. Ciò evita di dover gestire problemi di sincronizzazione che si avrebbero usando un server e/o client multiprocesso, ed anche di non dover gestire i vari errori che spunterebbero con l'uso di socket non bloccanti. L'applicazione fa uso esclusivamente di socket TCP, per via dell'affidabilità richiesta dal tipo di servizio: trattandosi di un'applicazione per lo scambio di messaggi e condivisione di file, è opportuno che si abbia certezza riguardo l'arrivo dei pacchetti, UDP non riuscirebbe a soddisfare tali requisiti. Sia sul client che sul server, esistono strutture dati nelle quali sono salvati i socket descriptor ai quali, via via, ci connettiamo: nel client, è la struttura "socket descriptor", nella quale associamo ad un certo *username* la sua *porta* e il relativo *socket descriptor,* mentre sul server si tratta di "*utentiConnessi*" in cui salviamo le medesime informazioni. Altro vantaggio dei socket TCP sta nel comunicare la chiusura del socket: nel qual caso, i descrittori vengono rimossi dalle strutture dati precedentemente esplicate.

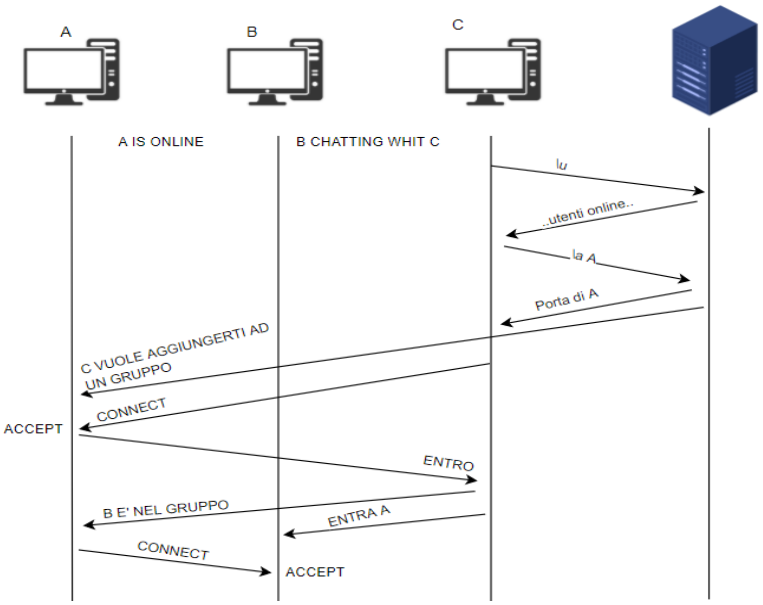
```
CODICE_HANGING = 1,
CODICE_SHOW = 2,
CODICE_CHAT = 3,
CODICE_SHARE = 4,
CODICE_OUT = 5,
CODICE_SENDUTENTIONLINE = 6,
CODICE_SENDUSERPORT = 7,
CODICE_CHATGRUPPO = 8;
```

Durante la comunicazione fra client e server, vengono usati spesso codici per identificare determinate azioni, nello specifico si tratta degli interi nell'immagine qui a sinistra. Se, per esempio, un utente esegue "out", andando offline, comunicherà il codice "5" al server, il quale avrà gli stessi codici e potrà riconoscere l'intenzione, e così via per tutti gli altri codici presenti nell'immagine.



CHAT: quando un utente vuole chattare con un altro, comunica la propria intenzione al server. Quest'ultimo, verifica se l'utente desiderato è online o meno, controllando il timestamp di logout della entry corrispondente a quello username. Nel caso in cui non lo fosse, ciò viene comunicato all'utente richiedente, e la chat prosegue fra l'utente e il server, il quale bufferizza i messaggi in arrivo. Sul server è presente una struttura dati che indica quali utenti sono in chat con un utente offline, in modo da sapere quali messaggi vanno bufferizzati e quali, invece, sono comandi. Se, invece, l'utente risulta online, il server comunica al mittente la porta per contattare l'utente cercato, ed il client stabilisce con esso una connessione. In questo caso, dopo l'avvenuta connessione, la chat prosegue esclusivamente fra i due utenti, senza altri interventi del server.

E’ inoltre presente un file di hanging per ogni ricevitore: nel momento in cui il server bufferizza un messaggio destinato ad un certo utente, apre il file di hanging relativo a quell’utente e salva il mittente, il timestamp attuale, ed in più incrementa il contatore relativo ai messaggi pendenti relativi a quel mittente. Tenendo in memoria quel file, ed aggiornandolo ad ogni arrivo di messaggi destinati ad utenti offline, si riesce ad evitare di calcolare, ogni qualvolta venga richiamata la hanging, tutti i dati da inviare. Il meccanismo è analogo a quello delle materialized view viste nel corso di Basi di Dati al primo anno.



CHAT DI GRUPPO: quando un utente è all’interno di una chat, può aggiungere altri utenti digitando “\a username”. Nel caso, viene contattato il server, il quale comunica se ‘*username’* è online. In caso affermativo, manda all’utente dentro la chat la porta di *username* ed a *username* il nome di chi vuole contattarlo. Nel caso in cui *username* non sia già all’interno di una chat di gruppo, accetta la richiesta. A questo punto, a tutti gli utenti già all’interno del gruppo viene comunicato, dal client che ha effettuato l’aggiunta, che *username* sta per entrare, mandandone la porta mentre a *username* vengono mandati i nomi degli utenti già presenti nel gruppo (funzione *sendGroupPartecipant*) , in modo da potersi connettere. Nel caso in cui un utente disponga già del socket descriptor relativo all’utente che sta per entrare nel gruppo, non si crea una nuova connessione, ma ci limitiamo a recuperare quello. Da quel momento in poi, ogni messaggio mandato da uno degli utenti nel gruppo è ricevuto da tutti gli altri, e la lista dei partecipanti al gruppo è a video. E’ presente una struttura dati in cui sono registrati gli utenti all’interno del gruppo. Se un utente digita “\q”, esce dal gruppo e gli altri utenti all’interno della chat ne vengono a conoscenza; i messaggi successivi, dentro il gruppo, non riguarderanno più l’utente uscito.

SHARE FILE: il protocollo per il file sharing coinvolge esclusivamente i peer, e si può attuare se l’utente è dentro una chat, che sia singola o di gruppo. Nel primo caso, il file è inviato un chunk per volta, con chunk grandi 128 byte, al singolo utente nella chat. Altrimenti, ogni chunk è mandato a tutti gli utenti della chat di gruppo. Si utilizza la funzione *share_file.* Prima dei chunk, è mandato (al singolo o al gruppo), il codice ARRIVE_FILE, relativo appunto all’arrivo di un file: un client che recapita tale stringa entra nella funzione *recv_file*, riceve i chunk uno alla volta e ricompone il file.

SHOW: quando un utente manda il comando di “show”, si controlla prima di tutto che siano presenti messaggi pendenti fra l’utente richiedente e l’utente specificato nel comando. In caso affermativo, si mandano i messaggi pendenti, recuperandoli dal relativo file. Sul server, per ogni utente destinatario è presente una cartella, con un file per ogni utente mittente. Quando “A” digita “show B”, il server controlla se nella cartella dei messaggi pendenti di “A” è presente il file “B.txt”, e nel caso manda tutti i messaggi pendenti (presenti nel file) ad “A”, aggiornando anche il relativo file di hanging di “A” (ovvero eliminando il record relativo a “B”). La divisione dei messaggi in più file serve ad evitare di scorrere, ad ogni richiesta di show, file di grandi dimensioni.