UNIVERSITÀ DI PISA


**Relazione del progetto di Reti Informatiche**

**A.A. 2021/2022**

Il progetto come da specifiche propone lo sviluppo di un'applicazione distribuita di instant messaging basata su un modello ibrido peer-to-peer e client-server.

Il sistema di chatting è stato implementato in due file: Server.c e Device.c. In Server.c è stato implementato il server mentre in Device.c è stato implementato il device.

Il sistema viene avviato eseguendo il comando bash exec.sh che avvia quattro terminali: uno per il server e tre per i device.

Ad ogni device, al momento dell'avvio, viene richiesta la porta del server al quale si vogliono connettere. Una volta connessi server possono registrarsi all'app, oppure, se hanno già un account, possono effettuare il login.

Il server mantiene le informazioni di ogni dispositivo registrato in un vettore (della dimensione del massimo numero di dispositivi registrabili). Ogni elemento del vettore è una struttura dati rappresentante un device. L'i-esima entrata rappresenta il dispositivo con id=i (id che viene assegnato al device al momento della registrazione).

## File e Directory

La directory ***rubric*** contiene i file con la rubrica di ogni dispositivo.

La directory ***server*** contiene due file, e una directory utili al server. Il file ***registro.txt*** è l'implementazione del registro richiesto nelle specifiche. il file ***usr_pwd.txt*** contiene stringhe del tipo "usernamepassword", viene aggiornato in fase di registrazione di un dispositivo e utilizzato in fase di login per verificare se la password inserita è corretta o meno. La directory ***pending_messages*** contiene a sua volta tante directory quanti sono i device registrati, ognuna nominata con l'username del dispositivo. Queste cartelle contengono file sui quali vengono salvati i messaggi pendenti ricevuti dagli altri utenti.

Per ogni utente registrato viene creata una directory nominata con il suo username. Ognuna di queste contiene i log delle chat con gli altri utenti. Un file, nominato con l'username dell'altro utente, per ogni chat. Inoltre, se l'utente ha effettuato il logout mentre il server era offline, salva su un file nominato ***logout.txt*** (all'interno della "propria" directory) il proprio timestamp di logout.

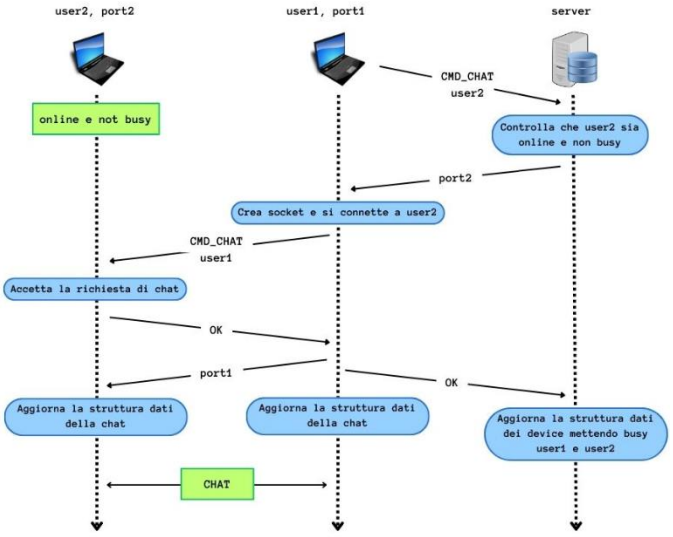
Il file ***server_recover.txt*** è un file di backup, che viene creato dal server al momento della disconnessione. In questo file vengono salvati i dati importanti degli utenti registrati. Al momento del nuovo avvio così il server può recuperarli.

La directory **files** contiene due file di testo utilizzati per testare lo share di essi nella chat.

## Avvio di una chat

Gli utenti possono avviare chat con gli altri utenti che hanno in rubrica. L'avvio è rappresentato sinteticamente in figura (caso in cui dev2 sia disponibile).

Nel caso in cui il dispositivo con il quale un utente vuole iniziare la chat sia offline o occupato i messaggi inviati verranno salvati dal server e poi recapitati quando il destinatario lo richiederà. Nel momento in cui il destinatario leggerà i messaggi 'pendenti' il mittente riceverà una notifica di avvenuta lettura.



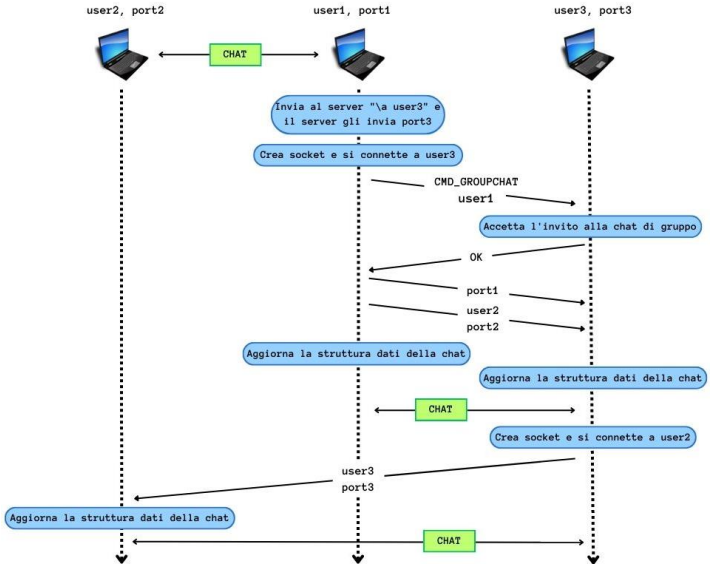

## Avvio di una chat di gruppo

I dispositivi in chat possono poi aggiungere altri utenti in modo da creare una chat di gruppo.

Nella figura a lato è rappresentata l'aggiunta di un utente ad una chat che passa da 2 a 3 utenti (è omessa la richiesta di dati che user1 fa al server riguardo a user3.

Ad ogni aggiunta, ogni coppia di dispositivi sarà connessa da un socket di comunicazione: in questo modo i dispositivi possono uscire dalla chat indipendentemente dagli altri.

Il problema di questa soluzione è la scalabilità all'aumentare dei dispositivi ( [n*(n-1)]/2 connessioni con n numero di dispositivi in chat ).




## Gestione di una chat

Ogni partecipante della chat memorizza le informazioni riguardanti gli altri membri in un vettore, i cui elementi sono strutture dati che contengono le informazioni utili di ogni altro utente in chat: username, porta e descrittore di socket con cui sono collegati.

Il formato dei messaggi è: ** **[username] [timestamp] text** (per i messaggi inviati e non recapitati all'inizio è presente solo un ***).

Se la chat è solo tra due utenti viene mantenuta la cronologia dei messaggi inviati/ricevuti da entrambi gli utenti. Entrambi gli utenti avranno la cronologia salvata su file, che vengono aggiornati al momento dell'invio/ricezione di ogni messaggio. La cronologia non è salvata per le chat di gruppo.

Per quanto riguarda lo share dei file, l'utente che effettua la condivisione apre il file che vuole condividere e memorizza in una variabile il contenuto. Prima di tutto invia ai partecipanti il nome del file condiviso e successivamente il contenuto. Una volta avvenuto lo share del file, i membri della chat avranno nella cartella personale il file interessato, mentre nella chat verrà stampato il nome del file e il suo contenuto.

### Gestione delle disconnessioni improvvise

Le disconnessioni improvvise di un device o del server sono gestite tramite l'utilizzo della libreria signal.h che permette di implementare degli handler che si attivano nel momento in cui il programma riceve un determinato segnale. Sono stati implementati gli handler per la gestione del SIGINT (CTRL+C da terminale) e SIGTSTP (CTRL+Z da terminale). Gli handler fanno sì che la disconnessione improvvisa venga gestita come una disconnessione volontaria, così da non perdere nessuna informazione.

### Gestione delle connessioni

Le connessioni instaurate nel sistema seguono tutte il protocollo TCP e la conseguente affidabilità. Il Server gestisce le richieste in modo concorrente; per ogni Device online nel sistema viene creato un thread che gestisce le sue richieste. Le primitive utilizzate sono la pthread_create() e la pthread_exit(), quest'ultime sono state preferite alla fork() e alla exit() poiché comportano un overhead più basso e una gestione più efficiente della concorrenza. Per l'implementazione di questa politica è stata inclusa la libreria pthread.h ed è stata aggiunta l'opzione -lpthread al comando di compilazione del Server.

***Matteo Lombardi***