Relazione sul progetto chatterbox

Gioele Bertoncini

1 Struttura del server

1.1 Strutture dati

1.1.1 Tabella degli utenti

La struttura principale che tiene traccia delle informazioni sugli utenti del server è definita dal tipo *users_table_t* definita e implementata nei file *users_table.h* e *users_table.c*. Al suo interno memorizza i dati relativi a gruppi e utenti in due tabelle hash (la cui implementazione si trova nei file *icl_hash.h* e *icl_hash.c*, entrambi forniti durante il corso). La prima tabella hash contiene elementi di tipo *chat_user* che modellano un singolo utente registrato, composti da un array di caratteri per rappresentare il nickname, un intero che indica il numero del file descriptor su cui l'utente attualmente connesso e una coda dei messaggi ricevuti dall'utente (vedi 1.1.2). La seconda tabella contiene la lista dei gruppi creati dagli utenti, rappresentati dalla struttura *chat_group* che è composta da due array di caratteri (nickname del proprietario e nome del gruppo) e dalla lista dei membri del gruppo (vedi 1.1.3).

La tabella degli utenti fa uso anche di un array di n+m+f mutex per sincronizzarsi rispettivamente su utenti, gruppi e descrittori di file, l'indice con cui vi si accede è calcolato applicando la stessa funzione usata per le tabelle hash modulo n, m o f (nel caso dei dei descrittori viene usato direttamente il valore) . Le prime n posizioni sono riservate alle mutex per gli utenti e le successive m a quelle per i gruppi, in questo modo sono maggiori le possibili di eseguire pi operazioni contemporaneamete. Le procedure per ottenere e rilasciare le lock sono implementate utilizzando alcune macro, in particolare *LOCKINORDER* e *UNLOCKINORDER* confrontano i valori di due nomi in modulo per acquisire/rilasciare le lock in ordine corretto, *LOCKALL* e *UNLOCKALL* acquisiscono/rilasciano tutte le lock dell'array in sequenza.

1.1.2 Code di messaggi, richieste e descrittori

Il server utilizza 3 tipi di code, tutte definite in *queue.h* e *queue.c* (i due file sono quelli forniti dai docenti durante il corso, modificati opportunamente togliendo la sincronizzazione interna e aggiungendo una funzione per liberare la memoria). Ogni utente (*chat_user_t*) ha una coda di messaggi che rappresenta la sua history, ogni thread worker fa uso inoltre di una coda di interi per ricordarsi i file descriptor a cui deve inviare i messaggi broadcast. L'ultima coda usata per inviare richieste dal thread *listener* ai *worker* e contiene elementi di tipo *request_t* composti da un messaggio e un descrittore di file.

1.1.3 Lista dei membri di un gruppo

Per ogni gruppo registrato sul server viene mantenuta la lista dei membri, implementata come linked list di stringhe dai file *string_list.h* e *string_list.c*. Per inviare messaggi o file ai membri di un gruppo si recuperano i loro nomi attraverso la funzione *getByIndex* che permette di accedere agli elementi della lista tramite indice. Per evitare di appesantire l'operazione di deregistrazione di un utente, quest'ultimo non viene cancellato immediatamente dai gruppi di cui era membro. Il controllo e l'eventuale rimozione viengono fatti durante l'invio di messaggi a un gruppo.

Questa lista è inoltre utilizzata per tenere traccia degli utenti online e dei rispettivi file descriptors, è necessaria nel caso in cui il thread *listener* (vedi 1.2) decide di chiudere una connessione ma conosce solo il file descriptor su cui l'utente è connesso e non il suo nickname.

1.2 Threads

Il server utilizza 1 thread per gestire i segnali *signal_handler* che si sospende con la *sigwait*, 2 thread per gestire le connessioni: il thread *dispatcher* che esegue in un ciclo infinito la accept e segnala ogni nuovo file descriptor al thread *listener* che esegue una select per trovare quali descrittori di file sono pronti per essere letti. Da questi vengono lette le richieste dei client che vengono inserite nella coda *pendingRequests*. Infine viene attivato un pool di thread *workers* i quali estraggono e elaborano le richieste e rispondono opportunamente ai client. I thread del server comunicano tra di loro attraverso una pipe dove i *workers* e il *dispatcher* producono file descriptors e il *listener* li consuma.

1.3 Chat Parser

Il main fa anche uso delle funzioni fornite da *chat_parser.h* e *chat_parser.c* per recuperare dal file di configurazione le informazioni necessarie per il funzionamento del server. In aggiunta a quelle già previste dal kit del progetto sono state aggiunte:

UserLocks	Numero di mutex utilizzate per sincronizzarsi sugli utenti
GroupLocks	Numero di mutex utilizzate per sincronizzarsi sui gruppi
FdLocks	Numero di mutex utilizzate per sincronizzarsi sui descrittori
Buckets	Numero di buckets per le tabelle hash

2 Terinazione del server

Quando il thread *signal_handler* riceve il segnale SIGUSR1 scrive un -1 sulla pipe del *listener*, quando quest'ultimo lo legge invia n messaggi con operazione OP_END sulla coda dove n è il numero dei thread *worker*, il thread *dispatcher* viene cancellato.

3 Sincronizzazione

Oltre alla sincronizzazione interna alla tabella degli utenti (vedi 1.1.1), i thread fanno uso di altre 2+n mutex: una per sincronizzarsi sulla pipe, una per sincronizzarsi sulla coda delle richieste e un array di n per sincronizzarsi sui descrittori durante le operazioni di write e read.

4 Script bash

Lo script bash si serve dei comandi cut e grep per selezionare il pattern corrispondene e di tr per eliminare spazi e tab. Inoltre utilizza il comando find per eliminare i file più vecchi.

5 Sviluppo del codice

Tutto il codice stato sviluppato utilizzando *gedit*, compilato e testato tramite il Makefile fornito nel kit del progetto.

6 Note

Nel file *icl_hash.c* nella funzione di eliminazione di un elemento il contatore degli oggetti contenuti nella tabella hash veniva incrementato anzichè ridotto. Nell'implentazione della tabella degli utenti c'è anche una funzione per l'eliminazione dei gruppi, non viene mai usata dai test.

Lo standard out e lo standard error del server sono ridirezionati su due file nella cartella */tmp*.