Relazione progetto Chatterbox
Laboratorio di Sistemi Operativi

Giuseppe Muntoni

Contents

1 Architettura del server **2**
- 1.1 Comunicazione tra thread listener e thread del pool 2
- 1.2 Terminazione del server 2
- 1.3 Task thread listener 2
- 1.4 Gestione dei segnali 3

2 Strutture dati **3**
- 2.1 Utenti registrati 3
- 2.2 Lista degli utenti connessi 3
- 2.3 History 3
- 2.4 Coda dei descrittori 4
- 2.5 Tabelle hash 4

3 Gestione della concorrenza **4**
- 3.1 Strutture dati 4
- 3.2 Descrittori dei client 4

4 Gestione dei file **5**

5 Testing **5**

1 Architettura del server

Il server Chatterbox presenta una struttura multithreaded. In particolare, il thread main crea un thread listener che si occupa dell'accettazione delle nuove connessioni e delle richieste da parte dei client. Crea inoltre un pool di threads, ciascuno dei quali si occupa di gestire singole richieste di un generico client.

1.1 Comunicazione tra thread listener e thread del pool

Ogni qualvolta arriva una richiesta da parte di un client, il thread listener inserisce il suo descrittore in una coda condivisa con i threads del pool e, fino a quando la gestione dell'operazione non è conclusa, non verranno accettate ulteriori richieste dal client. Uno dei threads del pool estrae il descrittore dalla coda, effettua la lettura della richiesta e la gestisce. Se l'operazione ha successo, attraverso l'ausilio di una pipe, il thread del pool invia il descrittore del client al listener in modo che questi possa riprendere ad accettare le richieste di tale utente. Se invece l'operazione fallisce il client viene disconnesso. In seguito, il thread del pool si mette in attesa di leggere un nuovo descrittore dalla coda.

1.2 Terminazione del server

Il thread main, una volta creati tutti i threads, si mette in attesa di terminare effettuando la join su tutti i threads. Il server può terminare per uno dei seguenti motivi:

- **Arrivo di un segnale:** il thread listener, appena si accorge dell'arrivo di un segnale, richiede a tutti i threads del pool la terminazione, attraverso un valore speciale inserito nella coda dei descrittori.
- **Fallimento di tutti i threads del pool:** durante la gestione di una richiesta, un thread del pool potrebbe fallire a causa di un errore di una chiamata di sistema o nella gestione della memoria dinamica. In tal caso, il thread decrementa un contatore globale dei threads del pool attivi e termina. Il listener non appena si accorge, accedendo al contatore, che non ci sono più threads del pool attivi, termina.
- **Fallimento del thread listener:** se fallisce una chiamata di sistema o la gestione della memoria dinamica, il thread listener richiede la terminazione di tutti i threads del pool e termina.

Una volta che tutti i threads terminano, il thread main si risveglia e, dopo aver effettuato la deallocazione di tutti le strutture dati, termina anch'esso.

1.3 Task thread listener

Il thread listener, il cui task è definito ed implementato nei file listener.h e listener.c, utilizza la chiamata di sistema select per mettersi in attesa finchè non arriva una richiesta di connessione o operazione da un client già connesso, finchè non è pronto il descrittore della pipe ed infine finchè non arriva un segnale.

1.4 Gestione dei segnali

La gestione dei segnali avviene tramite la chiamata di sistema signalfd. Signalfd restituisce un descrittore che risulterà pronto in lettura non appena arriva uno dei segnali presenti nel set passato come parametro. In tal modo, il thread listener può inserire nel set dei descrittori della select anche quello restituito dalla signalfd e risvegliarsi non appena arriva un segnale.

I segnali per i quali non viene lasciata la gestione di default sono:

- **SIGPIPE** che viene ignorato.
- **SIGUSR1** che viene ignorato se non è presente l'opzione **StatFileName** nel file di configurazione. Altrimenti viene mascherato e causa la stampa delle statistiche in un file file il cui path è specificato nel file di configurazione.
- **SIGINT, SIGTERM e SIGQUIT** che vengono mascherati e causano la terminazione del server.

2 Strutture dati

2.1 Utenti registrati

La struttura dati principale, che si occupa di supportare la gestione degli utenti registrati, è `users_t`, definita in users.h. Questa, mantiene una tabella hash che associa ad un nickname di un utente le sue informazioni ed i suoi dati. Inoltre mantiene un'ulteriore tabella hash che associa ad un descrittore di client il suo nickname. Quest'ultima risulta particolarmente utile in fase di disconnessione. Infine, salva il numero di utenti registrati e connessi.

Le informazioni ed i dati associati al singolo utente sono salvate nella struttura dati `users_data_t`, definita in `users_data.h`.

2.2 Lista degli utenti connessi

La struttura dati e le funzioni di supporto alla gestione della stringa degli utenti connessi sono definite in `users_list.h`. La stringa viene gestita in maniera dinamica. In particolare, ogni volta che si connette un nuovo utente, il suo nickname viene inserito in fondo alla stringa ed ogni volta che un utente si disconnette, si effettua una ricerca del nick nella stringa e lo si elimina mettendo al suo posto l'ultimo nickname.

2.3 History

Per la gestione della history è stato utilizzato un vettore circolare con politica FIFO e definito in boundedqueue.h. Tale struttura dati non è stata implementata interamente da me, ma è stata trovata nelle soluzioni di uno degli assegnamenti svolti durante il corso e modificata in maniera sostanziale in modo

da essere utilizzata per tale scopo. Inoltre è stata aggiunta la definizione ed implementazione di un iteratore.

Nel file `history_msg.h` è definito il formato dei messaggi inseriti nella history.

2.4 Coda dei descrittori

La coda dei descrittori utilizzata da thread listener e threads del pool è definita nel file queue.h. Questa è unbounded e incapsula la gestione della concorrenza. Così come per il vettore circolare, anche tale struttura dati non è stata realizzata interamente da me ma è stata trovata nelle soluzioni di uno degli assegnamenti di laboratorio e successivamente modificata.

2.5 Tabelle hash

Le tabelle hash utilizzate sono definite nel file `icl_hash.h`. Non sono di mia creazione ma è stato aggiunto un iteratore per poter iterare su tutti gli elementi della tabella hash.

3 Gestione della concorrenza

3.1 Strutture dati

Le funzioni che manipolano le strutture dati `users_t` e `users_list_t` non sono thread-safe. Per ogni campo della struttura dati sono state definite una funzione di lock e unlock per eseguirne in mutua esclusione la manipolazione. Questa scelta è stata fatta in maniera tale da poter utilizzare tali interfacce anche in una versione singlethreaded del server.

Le tabelle hash in `users_t` possono avere un'unica mutex oppure possono essere suddivise in blocchi logici ad ognuno dei quali è associata una mutex differente. In quest'ultimo caso, se un thread volesse manipolare un elemento all'indice i della tabella ed un altro thread volesse manipolare un elemento all'indice j e se questi fossero in blocchi logici differenti, allora potrebbero accedere contemporaneamente alla tabella hash senza che l'uno debba attendere l'altro.

Nell'implementazione del server è stato utilizzato l'approccio della suddivisione in blocchi logici in quanto può aumentare notevolmente il livello di concorrenza, dato l'elevato numero di accessi effettuato a tali tabelle durante la gestione delle operazioni.

3.2 Descrittori dei client

Ad ogni utente, al momento della registrazione, viene associato un id immutabile e non necessariamente unico utilizzato per indicizzare un array di mutex. In

particolare, la modifica del descrittore associato ad un utente, le write sul descrittore e la chiusura del descrittore vengono effettuate in mutua esclusione sulla mutex di indice id nell'array di mutex.

4 Gestione dei file

Per la gestione dei file sono state utilizzate le funzioni della librearia standard, in quanto tipicamente più efficienti e veloci rispetto alle chiamate di sistema POSIX.

Ogni volta che viene effettuata una richiesta di tipo POSTTXT ed esistono già 'n' file con lo stesso nome, allora viene creato un nuovo file al cui nome viene concatenato l'indice 'n+1'. Ci possono essere massimo 128 file con lo stesso nome. Per ogni utente, si tiene traccia in una tabella hash dei nomi dei file che gli sono stati inviati, in maniera tale che quando viene effettuata una GETFILE si possa verificare che il file richiesto sia effettivamente destinato a tale utente.

5 Testing

Il testing del codice è stato effettuato utilizzando le stampe sullo standard output e tool quali gdb e valgrind. La macchina fisica utilizzata per i test è dotata di cpu AMD Ryzen 5 2600 (6 cores e 12 threads) e OS Linux Ubuntu 18.04.1 LTS.