Progetto LSO: Chatty

Lorenzo Beretta

Marzo 2019

1 Architettura del Progetto

1.1 Struttura generale

Il main del server *Chatty* esegue per prima cosa il parsing del file di configurazione, dopo di che inizializza le strutture condivise tra tutti i thread, crea *ThreadsInPool* threads *worker* ed un thread *listener* ed infine si dedica alla gestione dei segnali mentre i threads avviati soddisfano le richieste dei clients.

- I threads **Worker** processano le richieste dei clients estraendole dalla coda condivisa *queue*.
- Il thread **Listener** rimane in ascolto di richieste di operazioni o nuove connessioni da parte dei clients e le inserisce nella *queue*.

1.2 Comunicazione tra Clients e Server

I Clients si connettono al server attraverso un socket linkato al path *UnixPath* impostato attraverso il file di configurazione. Una volta aperta la connessione i dati vengono scambiati scrivendo su un *file descriptor* riservato al client che lo ha aperto fintanto che questo rimane connesso. Questi scambi avvengono sfruttando le funzioni documentate in **connections.h**.
I file descriptors sono assegnati nel seguente modo:

- 0, 1, 2: come di default (*stdin, stdout, stderr*)
- 3: il socket per accettare nuove connessioni

- 4, 5: read e write end della pipe utilizzata per le comunicazioni interne
- 6, .. *MaxConnections* + 5: fd per la comunicazione con i clients.

Questa modalità ci permette di tenere sotto controllo il numero di file descriptors utilizzati affinchè non venga superato il limite di *MaxConnections* connessioni simultanee come impostato nel file di configurazione.

1.3 Strutture dati Condivise e Sincronizzazione

Ci sono varie strutture dati condivise, elenchiamole:

Statistiche: per la gestione delle statistiche esiste la variabile globale *sset* che viene aggiornata da tutti i thread dall'opportuna funzione update_stat (resa MT-safe dall'utlizzo di un mutex condiviso) docuementata in **stats.h**.

Coda: la variabile globale *queue* viene utilizzata per gestire le richieste dei clients in attesa di essere processate da un thread worker. La struttura è sincronizzata secondo il tipico schema produttore-consumatore utilizzando un mutex e una variabile di condizione, tutta la documentazione si può trovare in **queue.h**.

Dizionario: implementato attraverso una tabella di hash (per garantire le performances) sincronizzata utilizzando un mutex, serve per salvare i Nicknames registrati, i file descriptor ad essi assegnati se questi sono connessi e la loro history. Ho scelto come size 10^5 in modo da garantire un buon load factor. Nel codice corrisponde alla variabile globale *htab*. La documentazione si può trovare in **hashtable.h** e **icl_hash.h**.

Nickname Connessi: gli array *fd_to_nick* e *connected_fd* sincronizzati con il mutex *connected_mutex* servono rispettivamente a salvare la mappa inversa a quella fornita dalla hash table (che associa ad un nick il suo fd) e alla comunicazione worker-listener per segnalare se un client è ancora connesso o meno (vedere la sezione successiva). Questi sono documentati direttamente in **chatty.h**.

Esiste inoltre l'array *mtx_arr* che contiene MaxConnections + 6 mutex per sincronizzare la lettura e scrittura sui fd dei clients. Questo è necessario in

quanto un client può ricevere contemporanemente messaggi da utenti diversi e questo genera una race condition.

1.4 Comunicazioni Interne

Le comunicazioni interne sono le seguenti:

Listener-Worker: gestita attraverso la coda *queue*, il listener passa ai threads worker i fd dei clients da servire.

Worker-Listener: i workers comunicano al listener i fd dei clients che non si sono disconnessi. Questo è necessario perchè la disconnessione del client non è effettuata con un'operazione ma semplicemente chiudendo il fd (e comunque questo potrebbe avvenire in caso di disconnessione accidentale del client). Il listener deve quindi sempre supporre che un client si disconnetta una volta messo in coda ed eliminarlo dalla bitmask della select, altrimenti la select potrebbe essere destata in modo inconsistente. Il worker testa la disconnessione sfruttando la read con la quale legge il messaggio che il client utilizza per richiedere la sua operazione e se questo è ancora connesso lo notifica al listener. Per fare ciò viene utilizzata una pipe sulla quale scrivere messaggi fittizi che servono a destare la select del listener, senza questo accorgimento il listener potrebbe rimanere bloccato nella select quando avrebbe ancora da ascoltare un client.

Main-Worker: nel caso in cui il processo riceva un segnale di interruzione (*sigint, sigquit o sigterm*) il main lo gestisce pushando nella coda *ThreadsInPool* volte il valore "-1". Questi fanno terminare l'esecuzione dei threads worker evitando che rimangano bloccati sulla wait nella *pop_queue*.

Main-Listener: analogamente per evitare che il listener si blocchi sulla select quando deve terminare a seguito della ricezione di un segnale il main scrive sulla pipe interna un messaggio fittizio.

2 Suddivisione del codice

Il codice del main e quello del listener si trovano in **chatty.c** mentre quello del worker in **worker.c**. Ho preferito suddividere così il codice in quanto en-

trambi costituivano due parti indipendenti (nel senso che utilizzano in larga parte procedure "proprie") e corpose del codice.

Il resto del codice è altamente modulare ed è quindi stato suddiviso (in accrodo con quanto fornito alla consegna) in file **.h** (contenenti la documentazione) e file **.c**. I moduli sono i seguenti:

- **connections**: contiene le funzione che implementano la comunicazione server-client.
- **queue**: libreria per la gestione della coda condivisa.
- **nickname**: tipo di dato della hash table.
- **icl_hash**: libreria per una hashtable generica.
- **hashtable**: libreria per gestione dell'hash table che associa ai nicknames i loro fd e la history.
- **ops**: definisce i codici delle operazioni utilizzate nei messaggi tra client e server.
- **message**: contiene una libreria che gestisce la formattazione dei messaggi per le comunicazione client-server.
- **stats**: implementa le funzioni per la gestione delle statistiche.
- **parser**: implementa la funzione che effettua il parsing del file di configurazione.
- **config**: (file fornito dai docenti) contiene la definizione di alcune costanti (come ad esempio il numero di buckests della hash table).

3 Test

Ho testato il server chatty sul mio laptop personale: Debian 9, processore i3-6006U (x64) , 4 cores.

Durante la stesura del codice ho testato indipendentemente i vari moduli, i test sono stati effettuati con programmi che ho prodotto sul momento e,

una volta accertata la funzionalità del codice, eliminato. Non ho ritenuto necessario riscrivere dei test per parti che sapevo funzionare una volta che il server dimostra di superare i test forniti dai docenti.

Prima di provare i test forniti dai docenti ho eseguito il server scambiando messaggi manualmente tra vari clients aperti in diversi terminali, facendo ciò ho testato preliminarmente tutte le operazioni implementate da *chatty*.

Ho infine eseguito con successo tutti i tests forniti dai docenti.