Relazione progetto di Sistemi Operativi e Laboratorio

Fausto F. Frasca - 559482

A.A 2018/19

Contents

1 Strutture dati di appoggio

1.1 Tabella hash

La tabella hash (*hash.h*, *hash.c*) contiene tutti i nomi dei client online e le collisioni sono gestite con liste di trabocco.
Si è preferita tale scelta per la rapidità garantita in media dalla tabella.

1.2 Statistiche

1.2.1 Server

Per le statistiche del server si fa uso della struct *statistics_t* (*server_lib.h*) protetta dalla mutex *mutex_stat* (*server.c*). Con tale struct si tiene conto di:

- Client online
- Dimensione della cartella data
- Numero di oggetti salvati in data
- Istante in cui è stato avviato il server
- Client totali che si sono connessi al server

Si fa uso invece di una variabile *volatile sig_atomic_t (server_lib.c)* per tener conto di quanti byte vengono scambiati tra client e server.
Tale scelta è stata fatta per via del fatto che sig_atomic_t granatisce che tutti i thread leggono il medesimo valore, evitando così di dover usare una mutex.
Poichè tale variabile va aggiornata per ogni singola lettura/scrittura sulla socket si è notato che, dichiarandola in statistics_t, le eccessive lock/unlock su mutex_stat peggioravano le prestazioni.

1.2.2 Client

Per le statistiche del client si fa uso della struct *statistiche_t* (*client.c*). Con tale struct si tiene conto di:

- Test effettuato (1 store, 2 retrieve, 3 delete)
- Se connect e disconnect sono andate a buon fine
- Numero di store andate a buon fine e non
- Numero di retrieve andate a buon fine, non andate a buon fine e retrieve i cui dati ricevuti non corrispondono a quelli inviati
- Numero di delete andate a buon fine e non

2 Server

L'eseguibile *server* viene compilato linkando *server.o* e la libreria statica *libServerLib.a*, quest'ultima costituita da *server_lib.h* col corrispettivo file oggetto (2.2) e *hash.h* col corrispettivo file oggetto (1.1).

2.1 Server.c

Una volta avviato l'eseguibile, il server effettua una serie di azioni prima di creare la socket e mettersi in ascolto dei client tra cui: azzerare le statistiche, mascherare alcuni segnali, generare il thread handler e creare la directory `data`.
Il server fa uso di una select non bloccante, in modo tale da evitare di bloccarsi sulla accept e poter così controllare periodicamente la variabile globale *stop*, settata eventualmente dal thread handler (2.1.2).
Se la variabile risulta settata a 1, il server attende che tutti i thread attivi terminino. L'attesa viene interrotta non appena la variabile globale *thread_conn*, che tiene conto del numero di thread attivi, diventa 0.
Sia stop che thread_conn sono di tipo volatile sig_atomic_t.
Per quest'ultima è stato scelto tale tipo per motivi analoghi a quelli riportati nella sezione 1.2.1.

2.1.1 Thread worker

Tale thread viene generato non appena un client fa la connect sulla socket.
Si occupa di servire un solo client e, prima di registrarlo e inserire il suo nome nella tabella hash (1.1), si accerta che non ci sia già un altro client connesso che abbia il medesimo nome.
Si appoggia alle funzioni di supporto presenti nel file *server_lib.c* (*2.2*) per leggere i messaggi ricevuti dal client e capire quale operazione effettuare.
Se l'operazione risulta "malformed" vuol dire che il messaggio ricevuto non ha riespettato il protocollo di comunicazione stabilito. Come conseguenza l'fd per la comunicazione sulla socket viene chiuso e il thread termina.
Medesima conseguenza si ha quando l'operazione risulta "break". In questo caso vuol dire che la read sulla socket ha restituito 0.

2.1.2 Thread handler

Il thread handler viene generato dal server prima di mettersi in ascolto.
Gestisce i seguenti segnali: SIGINT, SIGQUIT, SIGTSTP, SIGTERM, SIGALRM e SIGUSR1.
Quest'ultimo segnale fa stampare le statistiche del server riportate nella sezione 1.2.1.
Gli altri invece settano la variabile globale *stop* a 1 facendo così terminare prima tutti i thread e dopo il server, come descritto sopra (2.1).

2.2 Server_lib.c

In questo file sono presenti tutte le funzioni ausiliarie necessarie per il corretto funzionamento del server. I prototipi delle funzioni sono riportati in server_lib.h.
Di seguito viene riportata la descrizione solo delle funzioni più particolari, per le altre fare affidamento ai commenti riportati nel file.

`raw_msg` ha il compito di leggere il messaggio del client e scomporlo, ricavandone l'operazione da effettuare e gli evenutali *name, len* e *data*. Restituisce al chimante tutte le informazioni acquisite ritornando una variabile di tipo *messaggio_t* (*server_lib.h*).
Se, durante la lettura dell'header, non vengono rispettate le condizioni stabilite nel protocollo di comunicazione o viene richiesta una STORE con len minore o uguale a 0, l'operazione da effettuare è settata su "malformed" e in data viene riportato il messaggio da restituire al client.
Se la read sulla socket restituisce 0, al chiamante viene ritornata l'operazione "break".

`create_directory` crea una directory con i permessi di read, write e execute per owner, group e others. Ritorna 1 se l'operazione è andata a buon fine, 0 altrimenti.

`store` apre il file dichiarato in *file_name* in sola lettura. Quest'ultimo viene creato se non è ancora presente nella directory, con permessi in lettura e scrittura, oppure viene sovrascritto se è già presente. Successivamente i dati presenti in block vengono trascritti sul file. Ritorna il messaggio da restituire al client.

get_time ritorna il tempo in millisecondi. Viene utilizzata per calcolare il delta tra l'avvio del server e quando viene inviato il segnale SIGUSR1, in modo tale da sapere da quanto tempo il server è attivo.

3 Client

L'eseguibile *client* viene compilato linkando *client.o* e la libreria statica *libClientLib.a* , quest'ultima costituita da *client_lib.h* col corrispettivo file oggetto (3.2).

3.1 Client.c

Il client accetta come parametri il nome da utilizzare nella connessione e il numero del test da eseguire (1 store, 2 retrieve, 3 delete). Una volta terminato, da in output le statistiche riguardanti l'esito delle operazioni effettuate per quel determianto test.
Fa uso della funzione *create_block* per creare il blocco di dati da passare sia al server per la store, sia alla funzione *check_retrieve* per assicurarsi che i dati

passati corrispondano.
Come richiesto, la grandezza dei dati va da 100 byte a 100000 byte.

3.2 Client_lib.c

Questo file va a costituire la libreria per la comunicazione con il server. Contiene tutte le funzioni richieste sfruttabili dal client.
L'unica funzione di supporto alle altre è *read_write_socket.*
La funzione ha il compito di inviare al server in un unico blocco il messaggio ricevuto e restituire la risposta al chiamante. Sarà poi quest'ultimo a gestire eventuali successi o errori.

4 Altri file

4.1 Makefile

Il makefile riporta tutti i target richiesti, oltre a quelli necessari per la compilazione dei vari eseguibili e librerie.
Una volta lanciato il target *test* viene eseguito prima il target *cleandata,* che elimina la cartella `data` e il file di log se presenti nella working directory. Successivamente viene lanciato, in maniera sequenziale, il server e lo script *start_test.sh.*

4.2 Start_test.sh

Lo script lancia i 100 client nel modo richiesto. L'output di ogni client (3.1) viene reindirizzatto nel file testout.log.
Quando tutti i client terminano, viene lanciato lo script *testsum.sh.*

4.3 Testsum.sh

Lo script legge il file di log e ne estrapola le informazioni da dare in output. Le informazioni riportate sono le seguenti:

Store — numero di client che hanno fatto la store, store totali, store andate a buon fine, store fallite, percentuale di store avvenute con successo.

Retrieve — numero di client che hanno fatto la retrieve, retrieve totali, retrieve andate a buon fine, retrieve fallite, retrieve i cui dati ricevuti non corrispondevano ai dati passati, percentuale di retrieve avvenute con successo.

Delete — numero di client che hanno fatto la delete, delete totali, delete andate a buon fine, delete fallite, percentuale di delete avvenute con successo.

Connect numero di client che hanno fatto la connect, connect totali, connect andate a buon fine, connect fallite, percentuale di connect avvenute con successo.

Disconnect numero di client che hanno fatto la disconnect, disconnect totali, disconnect andate a buon fine, disconnect fallite, percentuale di disconnect avvenute con successo.

Totale totale client lanciati, totale operazioni effettuate, totale operazioni andate a buon fine, totale operazioni non andate a buon fine, totale retrieve errate, percentuale di operazioni andate a buon fine

Una volta terminato manda al server prima SIGUSR1 e dopo SIGTERM.

4.4 Conn.h

Questo file contiene le funzioni *readn* e *writen* ed è usato sia da server.c e server_lib.c che da client_lib.c.
Le due funzioni sono le medesime utilizzate durante gli esercizi del corso.
Sono definiti anche il nome della socket e la struct *msg_t*, usata in server_lib.c e in client_lib.c.