Progetto SOL 2016-2017: CHATTERBOX

Peparaio Giacomo mat.530822 corsoA

CONTENUTI:

1: Introduzione

Il codice del progetto si concentra soprattutto nei due moduli lib_utils (funzioni gestione utenti/gruppi) e threads(thread worker,listener, signal thread)
inizializzabili rispettivamente con le due funzioni init_utenti e server_start. All'avvio il main riempie i campi della struct configurazione parsando il file in input grazie alla funzione "parse" e solo in seguito fa partire il threadpool. I thread principali sono il listener e il worker: il primo accetta nuove connessioni e/o vede se c'è qualcosa da "leggere" in quelle già esistenti, mentre il secondo si occupa delle varie richieste richiamando le funzioni fornite da lib_utils. Le funzioni per le statistiche sono implementate nel file stats.c

2: Interazione client-server

La comunicazione avviene con le funzioni dell'interfaccia connections, che inviano messaggi interi, oppure solo header o data. Ogni campo di un generico messaggio viene spedito tramite una write sul descrittore con parametri : puntatore all'oggetto e size dell'oggetto; lo stesso vale per le read. È da notare che ogni messaggio inviato deve essere sempre scritto sul buffer con l'ordine:
OP→nome sender→nome receiver→size del buffer→buffer;
altrimenti la loro lettura sarebbe inconsistente. Per rendere atomico l'invio di un messaggio e impedire quindi che 2 thread del pool scrivano contemporaneamente su un descrittore ho utilizzato un array di mutex, dove ogni indice corrisponde ad un descrittore.

Le connessioni al server sono gestite dal thread listener che accetta nuovi client aggiungendone il descrittore ad un SET. Durante tutta l'esecuzione di chatty il listener effettua la select e poi cicla sul SET per vedere quali descrittori(client) sono pronti a fare le loro richieste.

3: Interazione Listener-Worker

Ogni volta che il listener trova un descrittore "pronto" lo inserisce in una coda concorrente con politica FIFO e lo toglie dal SET della select. Questo descrittore verrà poi prelevato da uno dei thread worker che ne gestirà le varie richieste. Quando un worker ha finito di lavorare su un descrittore lo mette in un'altra coda concorrente(sempre FIFO) e "avverte" il listener che ha finito scrivendo su una pipe(il descrittore della pipe è nell'FD_SET della select), quest'ultimo si sblocca(era bloccato sulla select) e aggiorna il SET prelevando dalla coda. Quindi il ciclo degli spostamenti di un descrittore sarà il seguente:
LISTENER→CODA1→ WORKER→ CODA2→ LISTENER….ecc

4: Gestione utenti/gruppi

Ogni utente di chatty è rappresentato da una struct dove oltre ai campi nome,descrittore di connessione,ecc... è memorizzato anche il puntatore all'array dei messaggi della history.
Tutti gli utenti della chat(sia online che offline) sono memorizzati in un array gestito nel seguente modo: se ho una nuova registrazione cerco una cella libera ed inserisco, altrimenti rialloco. Ho scelto questa struttura dati ritenendo che il server è pensato per un uso ridotto(al massimo 1000 utenti) e , quindi, non si sarebbero intaccate le prestazioni scorrendo ogni volta l'array. Sempre per questo motivo ho optato per un unico mutex per l'accesso in mutua esclusione al vettore utenti.
Solo il thread worker interagisce con tale struttura grazie alle funzioni di interfaccia fornite da lib_utils(lo stesso vale per i gruppi), lui si occupa solo di leggere il msg dal descrittore e chiamare la rispettiva funzione in base alla OP letta.
Ogni gruppo, invece, è rappresentato da una struct con campi nome,creatore e array che contiene i puntatori ai nomi dei componenti.
Tutti i gruppi sono memorizzati in una tabella hash con chiavi:nomegruppo e data:struct gruppo. Anche qui, per l'accesso alla tabella in mutua esclusione mi avvalgo di una mutex.

5: Gestione segnali e statistiche

Tutti i segnali da gestire sono mascherati, la loro gestione è affidata al signal thread che, facendo la chiamata sigwait(con parametro la maschera dei segnali mascherati) in un loop capta un segnale, lo gestisce e poi si rimette in ascolto. Con SIGUSR1 il thread stampa la struttura chattystats(aggiornata dal worker in mutua esclusione con le funzioni di stats.h), invece se riceve un segnale di terminazione viene avvertita la select sempre con il metodo della scrittura su una pipe e il listener invierà un messaggio di terminazione a tutti i thread(tramite la coda).