Progetto SOL 2020/2021

Università di Pisa - Facoltà di Informatica

Jacopo Raffi 598092

Professore Alessio Conte

30 Settembre 2021

Indice

1 Introduzione

Il progetto consiste nella realizzazione di un $file\ storage$ gestito da un *server multithread*. I *client* comunicano con il *server* tramite un'apposita API.

2 Server

Il server immagazzina i $files$ in una *tabella hash* la cui dimensione può essere stabilita nel file di configurazione. La gestione della concorrenza avviene tramite delle opportune *mutex*, una per ogni lista della tabella.

2.1 Configurazione

All'avvio il server esegue il *parsing* del file di configurazione in formato *.txt*. Il $file$ deve avere la seguente struttura (anche in ordine diverso):
$numThread =< valore >$
$maxDim =< valore >$
$hashDim =< valore >$
$replace =< valore >$
In caso la struttura non venga rispettata il *server* viene avviato con parametri di $default$.

2.2 Threads

Il *server* è costituito da un numero variabile di *thread*. Il *thread manager* si occupa del *parsing* del $file$ di configurazione, della gestione dei segnali, della gestione del *socket* e della terminazione del *server*.
La comunicazione $manager - workers$ avviene tramite una *pipe* e una coda con politica $FIFO$, contenente i descrittori di $files$ associati ai vari *client*.
I $workers$ si occupano dello svolgimento delle richieste dei vari *client*.
Un *worker* si occupa di tutte le richieste di un singolo *client*, una volta terminata la connessione col *client* viene comunicata al *manager* la terminazone tramite la *pipe*.
Ogni operazione svolta dai $workers$ viene scritta su un apposito *file di log* (descritto di seguito).
La terminazion del *server* può avvenire tramite i segnali "SIGHUP", "SIGQUIT" e "SIGINT". Nel primo caso il *server* attende di completare le richieste dei *client* già connessi e non accetta altre connessioni. Nel secondo e terzo caso non si attende il completamento delle richieste ma si termina immediatamente. Per la gestione della terminazion e viene settata una variabile "end" a 1 in caso di terminazione immediata, viene settata a 2 in caso di terminazione "soft".

2.3 File di Log

Viene tenuta traccia di ogni richiesta eseguita dai $workers$ in un apposito *file di log*, in formato *.txt*. Ogni *worker* scrive il proprio *id thread*, l'operazione eseguita, il successo o meno dell'operazione ed eventuali altri parametri dipendenti dal tipo di operazione.

2.4 Politiche di rimpiazzamento

Nel server esiste la possibilità di avere delle *capacity miss*. Vengono implementate le politiche di rimpiazzamento $FIFO$ e LFU. La prima espelle i $files$ meno recenti in tempo $O(n)$, mentre la seconda si basa sul numero di operazioni eseguite sui vari $files$ ed impiega $O(nlog(n))$.

3 Client

Il *client* è un programma che accetta un numero di elementi da linea di comando e manda le relative richieste al server tramite la API fornita dal *server*. Ogni comando viene eseguito nell'ordine con cui viene passato come argomento del programma.

3.1 Gestione dei files espulsi

In caso di espulsione o di una lettura che implichi il salvataggio dei dati su disco il *client* può specificare da linea di comando in quali cartelle salvare i $files$ espulsi o i dati letti.

4 Makefile

Viene messo a disposizione un $Makefile$ che fornisce i $target$ $test1$, $test2FIFO$, $test2LFU$, $test3$ e $stat$. Tutti i test eseguono degli *script* nei quali viene avviato il *server* e successivamente vengono avviati i *client*.
Il primo test verifica che non ci siano dei *leak* in memoria tramite il comando $valgrind^{\smile} leak - check = full$.
I test $test2FIFO$ e $test2LFU$ verificano la correttezza delle rispettive politiche di rimpiazzamento. I $files$ espulsi sono memorizzati all'interno di apposite cartelle specificate dal *client*.
Il terzo test è uno stress test del *server*. Si considera superato se i valori statistici sono ragionevoli e se non sono presenti errori a $run - time$.

L'ultimo *target* "stat" prevede un sunto dell'esecuzione del *server*. In particolare :

- n. di *read* e *size media* delle letture in *bytes*;
- n. di write e size media delle scritture in *bytes*;
- n. di operazioni di *lock*;
- n. di operazioni di $open - lock$;
- n. di operazioni di *unlock*;
- n. di operazioni di *close*;
- dimensione massima in $Mbytes$ raggiunta dallo *storage*;
- dimensione massima in numero di file raggiunta dallo *storage*;
- numero di volte in cui l'algoritmo di rimpiazzamento della *cache* è stato eseguito per selezionare un file vittima;
- n. di richieste servite da ogni *worker thread*;
- massimo n. di connessioni contemporanee.

5 Note finali

Il progetto è stato reso pubblico su *github* al seguente *link*: `https://github.com/JacopoRaffi/StorageServer-Progetto-SOL`.