Progetto di Laboratorio di Sistemi Operativi

Rodrigo Casella 599523

1 Introduzione

Librerie terze parti

Nel progetto è stata utilizzata la libreria **icl_hash** fornita durante il corso, modificata con l'aggiunta di una struttura **icl_hash_iter_t** e delle funzioni **icl_hash_iterator_create**, **icl_hash_iterator_destroy** e **icl_hash_next** per iterare gli elementi delle strutture del tipo **icl_hash_t**.

Inoltre, è stata utilizzata l'implementazione di **boundedqueue** fornita durante il corso.

GitHub repository https://github.com/Rodrigo-Casella/file-storage-server

2 Filesystem

Mutua esclusione

Il Filesystem dispone di una mutex a livello globale che ogni thread deve acquisire prima di eseguire azioni che influenzano lo stato dell'intera struttura (e.g. la ricerca di un file, aggiungere un file). Inoltre, ogni file dispone di una propria mutex che i thread acquisiscono prima di modificare o leggere dati (e.g. scrivere/leggere i dati di un file, impostare flag).

Strutture dati

Il Filesystem mantiene i riferimenti ai file sia in una tabella hash che in una coda FIFO.

La tabella hash viene usata per le principali operazioni di ricerca e aggiunta file, mentre la coda FIFO viene usata per mantenere l'ordine di inserimento nel Filesystem e quindi per applicare le politiche di espulsione dei file.

Espulsione file

Il Filesystem espelle file in due casi: una richiesta di **openFile()** richiede l'aggiunta di una file, ma il server ha raggiunto il limite del numero massimo di file presenti sul server oppure una richiesta di **writeFile()** richiede la scrittura di più di dati di quanti il server può avere.

Sono state implementate le seguenti politiche di espulsione: FIFO, LRU, LFU e Second-Chance.

3 Server

Il Server va eseguito con il comando bin/server file-di-configurazione.txt

Manager

Il thread main del server, dopo aver letto il file di configurazione e aperto la comunicazione, si occupa di ricevere nuove connessioni, tramite **la select(),** e segnalare le richieste dei client ai thread worker tramite una coda thread-safe. I worker a loro volta segnalano al thread main il termine dell'elaborazione di una richiesta tramite una **pipe**, se il manager legge l'fd di un client significa che non ha terminato di inviare richieste e lo inserisce nuovamente nel **working_set** della **select**. Se invece legge 0 significa che un client ha terminato con le proprie richieste ed è uscito dal server.

Alla ricezione di **SIGHUP**, **SIGINT** o **SIGQUIT** il thread main non accetta più nuove connessioni e manda ai thread worker i messaggi di terminazione.

Worker

I thread worker ricevono dal thread manager l'fd dei client che richiedono una certa operazione sul server. In seguito, i worker mandano al client una risposta con l'esito dell'operazione richiesta e segnalano al manager se il client ha terminato o meno con le proprie richieste.

Logger

All'avvio del server viene lanciato un thread che si occupa di scrivere i messaggi di log sull'apposito file. I messaggi vengono letti da una coda thread-safe presente sul Filesytem.

Alla ricezione del messaggio di stop il thread logger chiude il file di log e termina.

4 Client

Il Client va eseguito con il comando bin/client -f socket-del-server [options].

I percorsi relativi delle directory mantenuti tali (e.g. opzioni –d e –D, opzione -w), mentre i percorsi relativi dei file vengono trasformati in percorsi assoluti.

Le richieste di scrittura non usano l'api **appendToFile()**.

5 API e Protocollo di comunicazione

Le richieste base del client usano il seguente protocollo di comunicazione:

Tipo di operazione (rappresentato da una variabile di tipo integer), Lunghezza della richiesta in byte, Richiesta (e.g. numero di file da leggere, path del file).

Per alcune operazioni vengono trasmessi dati aggiuntivi: i flag di apertura dei file nella **openFile()**, lunghezza del file e dati nella **writeFile()** e **appendToFile()**.

Il server risponde con un intero che rappresenta l'esito dell'operazione e con i dati dei file letti nel caso della della **readFile()** e **readNFiles()** o con i dati dei file espulsi nel caso della **writeFile()** e **appendToFile()**.

L'api **appendToFile()** è stata implementata, ma non è mai usata.