File Storage - Progetto di Laboratorio di Sistemi Operativi

sre990

28-03-22

Il progetto consiste nell'implementazione di un sistema di **file storage server**, realizzato mediante un singolo processo multi-threaded in grado di eseguire delle operazioni richieste da client multipli.

Le richieste vengono inviate dal client al server attraverso un'**API**.

E' presente un **Makefile** che consente all'utente dell'applicazione di generare gli eseguibili per il client e per il server tramite il comando *make all*. La directory di lavoro puo' essere ripulita da tutti i file e le cartelle generati nelle fasi di compilazione e di esecuzione con *make clean*.

L'app puo' essere testata con i comandi *make test1*, *make test2* e *make test3*, che creano ad-hoc alcune cartelle e file chiamati *stubX* utili al fine di eseguire i tre diversi tipi di test delle funzionalita' del file storage. Per ciascuno di questi test e' possibile, una volta finita l'esecuzione, stampare delle statistiche dettagliate sulle informazioni riportate dai file di log tramite i comandi *make stats1*, *make stats2* e *make stats3*.

Come parte opzionale per l'esame di Laboratorio di Sistemi Operativi sono state realizzate le politiche di rimpiazzamento nel caso di capacity miss note come **LRU** e **LFU**.

1 File Storage Cache

Lo spazio di memorizzazione del programma e' stato realizzato come una **cache** di file. In particolare, sono presenti due struct $cache\_t$ e $cache\_file\_t$ che rappresentano rispettivamente la cache e il file contenuto al suo interno.

La cache, al fine di manipolare i file e di fornire al server soluzioni per implementare tutte le operazioni del file system, si serve di una serie di funzioni, strutture dati e meccanismi.

Vi sono campi che memorizzano informazioni sulla dimensione e il numero di file, sia correnti che massimi; un campo per la scelta della politica di rimpiazzamento dei file; una **read-write lock** per permettere l'accesso alla struttura in mutua esclusione; una **doubly linked list** usata sia per la memorizzazione di file name all'interno della cache, sia di client che hanno aperto un determinato file; una **hashtable** con coppie chiave-valore costituite, rispettivamente,

dal nome di un file della cache e da un file di tipo $cache\_file\_t$, utile per realizzare operazioni di inserimento, ricerca e cancellazione in modo efficiente nella cache.

Per quanto concerne la struct $cache\_file\_t$, invece, sono disponibili, oltre ai campi ovvi come il nome, contenuto e dimensione, anche campi per realizzare le politiche di rimpiazzamento LRU e LFU che memorizzano la frequenza degli accessi al singolo file e il momento in cui l'ultima operazione sul file ha avuto luogo.

Infine, e' possibile realizzare l'accesso in mutua esclusione sul singolo file grazie a dei campi che forniscono una **read-write lock**.

2 Read-write Lock

Gli accessi in mutua esclusione possono avvenire sull'intero storage e sul singolo file. Si e' scelto di implementare il protocollo di mutua esclusione noto come **read-write lock**, dove e' consentito l'accesso concorrente in lettura a piu' lettori, mentre le operazioni in scrittura su un singolo file sono consentite a un solo scrittore per volta.

Per "operazioni di scrittura" a livello cache, si intendono tutte le operazioni che modificano il numero dei file, quindi ci si riferisce alle open con il flag O_CREATE settato, le write, le append e le remove. Mentre per quanto riguarda i singoli file all'interno della cache, le operazioni di scrittura sono tutte quelle che vanno a modificare i campi della loro struttura.

Ogni volta che si vuole accedere a un file per eseguire un'operazione su di esso, per prima cosa si acquisisce una lock su tutta la cache. Se l'operazione in questione e' di scrittura anche la lock corrispondente lo sara'. Se il file e' presente in cache, la lock verra' acquisita sul singolo file.

3 Server

Il server si occupa della gestione della cache in base ai dettagli letti da un file di configurazione. Nel file di configurazione sono specificati il percorso del socket a cui connettersi e del file per effettuare il logging; la politica di rimpiazzamento che deve essere seguita per le capacity miss; dettagli sulla capacita' della cache; il numero di **worker** presenti nel **pool**.

3.1 Worker

Ciascun worker viene incaricato dal server di gestire le richieste provenienti dal client (chiamate "*task*") secondo il paradigma master-worker.

I task vengono inseriti in un coda in **bounded buffer** insieme al file descriptor del client che ha richiesto l'operazione. Ciascun worker si occupa di recuperare i task, tradurli nelle operazioni appropriate specificate dalla API, eseguirle e infine di inviare il risultato dell'operazione al client tramite il suo fd e fare il logging dell'operazione.

4 Client

Il client si occupa di interpretare gli argomenti da linea di comando e, se questi sono validi e corrispondono a operazioni implementate, si servira' dell'API per fare richieste al server.

5 Semantica delle operazioni

Alcune scelte progettuali riguardano la semantica delle funzioni, con la conseguenza che certe sequenze di operazioni possono portare alla restituzione di un $OP\_FAILURE$. Ad esempio per quanto riguarda *lockFile* si e' previsto che la lock possa essere acquisita soltanto per i file che siano gia' stati aperti in precedenza dal client.

Per la *openFile*, non e' possibile per un client aprire un file che e' gia' aperto da esso. Mentre un file chiuso con *closeFile* deve prima essere stato aperto dallo stesso client.

Leggere un file con *readFile* e' un'operazione vincolata dalla openFile, che deve precederla.

Nella scrittura di un file con *writeFile* e *appendToFile*, puo' verificarsi un errore nel caso in cui il file da scrivere venga espulso prima del completamento dell'operazione.

6 Gestione degli errori

Gli errori si dividono in due tipi, quelli che restituiscono OP_EXIT_FATAL e quelli che restituiscono $OP\_FAILURE$. L'insorgenza del primo riguarda tutte le operazioni che lasciano la cache in uno stato inconsistente, mentre il secondo si verifica per errori semantici. Gli errori vengono fatti galleggiare fino al thread chiamante, che li gestira'. Per esempio, nel caso di un OP_EXIT_FATAL il client che lo riceve terminera' immediatamente l'esecuzione.

7 Logging

Per effettuare il logging delle operazioni effettuate viene usata la macro LOG_EVENT, che scrive l'operazione in mutua esclusione su un file di log, il cui path e' specificato nel file di configurazione.

8 Repository

Il codice di questo progetto e' disponibile all'indirizzo:

`https://github.com/sre990/sol-project`