Progetto SOL

Laboratorio Sistemi Operativi

Francesco Camaccioli
Matricola 597603

Storage

Le Strutture Dati utilizzate sono l' HashTable di Jakub Kurzak fornitaci a lezione e una coda di Filepaths, utilizzata per effettuare l'eviction FIFO mantenendo le chiavi dell' HashTable in ordine di inserimento. Quando sarà necessaria un'eviction, basterà semplicemente effettuare una pop dalla coda di Filepaths per poter trovare il file nell' Hashtable ed eliminarlo da questa. Ovviamente sia la coda che l' Hashtable sono aggiornate sempre contemporaneamente, per garantirne la consistenza.
Inoltre, nella struct Storage si mantengono i parametri massimali forniti dal configfile e i valori attuali della capienza dello storage stesso. Vi è infine presente una Mutex globale, che viene acquisita al momento della gestione di una richiesta e rilasciata opportunamente a secondo del tipo della richiesta ottenuta.
La struct File invece definisce un File all'interno dello storage. Esso avrà, oltre ai campi ovvi come path e contenuto, delle variabili intere che conterranno l'fd del client che è, rispettivamente, scrittore, creatore o locker, e una variabile che contiene il numero attuale di readers del file stesso. Vi sono inoltre presenti due FdList, lockwaiters e openers , delle code di file descriptor relativi ai clients, che conterranno rispettivamente la coda di clients che attendono di acquisire la lock sul file e la coda di clients che hanno aperto il file attraverso la openFile.
La mutua esclusione dei Files è garantita attraverso due Mutex e una Condition Variable, quest'ultima utilizzata nel caso di scrittura del file.

Server

Configurazione
La configurazione del server avviene attraverso la struttura configList, una coda contenente, in ogni suo nodo, una coppia name-value, opportunamente ottenute effettuando il parsing del file di configurazione passato al server da linea di comando. E' inoltre possibile introdurre commenti nel file di configurazione inserendo un '#' all'inizio del commento stesso.
Una volta ottenuti i parametri dal config, se questi non soddisfano le richieste (come ad esempio inserire un numero negativo come massimo numero di files nello storage) verranno inizializzati a valori di default, contenuti nel file utils.h.
Gestione delle connessioni e richieste
Le connessioni con i client vengono eseguite attraverso una socket e sono gestite da una select, in cui si distinguono 3 casistiche: nuova connessione, richiesta da un client già connesso e richiesta di chiusura connessione. Le richieste vengono inserite in una RequestQueue, che conterrà in ogni nodo l'fd del client richiedente. Questa coda di richieste è opportunamente dotata di variabili di gestione della concorrenza (una mutex e 2 condition variables) poiché è proprio da questa che i thread worker del server andranno ad estrarre l'fd del client da servire.
Nel processo main del server vengono infatti creati 'n' threads worker (in base al file di config), che andranno ad estrarre la prima richiesta della RequestQueue e la gestiranno a seconda del suo tipo. Terminata la gestione, il worker invierà al main thread del server l'fd del client che ha gestito attraverso una pipe. Nel caso in cui invece la richiesta è di uscita (REQ_EXIT), il worker chiuderà la connessione col client ed invierà la stringa "exit" nella pipe, la quale sarà letta dal main thread, che andrà rendere il worker libero di svolgere una nuova richiesta.
Gestione Segnali
Si gestiscono i segnali SIGHUP, SIGINT E SIGQUIT rispettivamente aggiornando delle variabili volatile sig_atomic_t utilizzate come guardia nel main loop del thread main del server. Si garantisce l'uscita istantanea (senza gestione delle richieste già presenti in coda) all'arrivo di un segnale SIGINT o SIGQUIT, mentre con un segnale SIGHUP si svolgeranno prima tutte le richieste e poi si terminerà il processo.
Log
Il file di Log (log.txt) viene scritto dagli Workers subito dopo l'invio della risposta alla richiesta client gestita dal singolo Worker, e la concorrenza delle scritture è garantita attraverso una Mutex.

Client

Il parsing dei command line arguments dei singoli clients viene effettuato attraverso getopt, inserendo le coppie opzione-argomento in una opportuna coda “OptList”.
Si prosegue tentando di aprire la connessione con la socket del server e, in caso positivo, si eseguono le chiamate API attraverso “makeAPiCall”, che riceve un nodo della coda OptList e in base al tipo di opzione andrà ad effettuare la chiamata API opportuna.
Inoltre, sono supportati sia path relativi che path assoluti.

Comunicazione e Client API

La comunicazione lato Client avviene attraverso l’invio di “Request”, una struct che definisce una generica richiesta, contenente tutte le informazioni necessarie a gestire le chiamate API. A seconda del tipo chiamata, l’API creerà una Richiesta opportuna e la invierà al Server. Il Server a sua volta, una volta gestita la richiesta di un client, invierà allo stesso una Response, che similmente conterrà le informazioni richieste dal Client (o eventualmente un codice di risposta negativo). Sia Request che Response sono contenute nel file comms.h; la prima conterrà un codice di identificazione del tipo (Open, Close, Exit, ecc…), le flags della Open, i parametri (l’N di ReadNFiles), e le informazioni relative al file richiesto. La seconda invece avrà un tipo (Ok, Notfound, Denied, …) e una fileList, utilizzata per inviare uno o più files al client, a seconda del tipo di Richiesta.
Non essendoci un opzione per effettuare una file append, la chiamata di “appendToFile” è stata resa possibile nel caso in cui si cerca di effettuare una writefile di un file già scritto: si andranno quindi a scrivere i ‘data’ richiesti, controllando se sono necessarie delle eviction per quantità in bytes.

Github

Link repository github: https://github.com/Testeena/SOLProject