Relazione del progetto del corso di Sistemi Operativi e Laboratorio A.A 2020/2021

Matteo Tolloso, matricola 598067

File di configurazione

Il path del file di configurazione viene specificato come primo ed unico argomento da riga di comando del server. Il formato è *chiave=valore;* per ogni riga. Le possibili chiavi settabili sono: *socket_name, num_thread_worker, max_dim_storage, max_num_file, log_file_path*. L'ordine con le quali vengono settate non è rilevante.

Thread main

Il thread main si occupa di inizializzare tutto il necessario al funzionamento del server, di gestire le richieste (manager) e di effettuare le operazioni di pulizia e chiusura:

- Installazione degli handler
- Parsing degli argomenti dal file di configurazione
- Inizializzazione della coda concorrente usata dal thread manager per distribuire ai thread worker i FD dei client che hanno fatto una richiesta da gestire
- Preparazione della pipe che serve ai thread worker per ritornare al thread manager i FD dei client che hanno completato una richiesta
- Preparazione della pipe che serve all'handler per comunicare alla select di aver ricevuto un segnale
- Inizializzazione del filesystem
- Creazione dei thread worker con tutti gli argomenti necessari
- Inizializzazione del socket e ciclo di manager in cui vengono ascoltati il socket, i client connessi e le due pipe
- Join dei thread e liberamento della memoria

Thread worker

I thread worker si occupano di eseguire le richieste attraverso un ciclo

- Pop di un client con una richiesta in attesa nella coda concorrente
- Esecuzione della richiesta
- Ritorno del FD al manager

File system

Il file system è in grado di gestire richieste concorrenti in lettura e scrittura attraverso un sistema di mutua esclusione a due livelli: sul filesystem e sul singolo file. Le ricerche e le modifiche nel filesystem vengono fatte in mutua esclusione tra tutti i worker, invece per quanto riguarda i singoli file viene usato un protocollo di tipo lettore/scrittore che permette letture concorrenti e scritture in mutua esclusione.
Le operazioni che modificano un file seguono lo schema: *lock filesystem, ricerca file, ingresso in scrittura file, modifica file, uscita in scrittura file, rilascio lock filesystem*
Le operazioni che leggono un file seguono lo schema: *lock filesystem, ricerca file, ingresso in scrittura file, rilascio lock filesystem, lettura file, uscita in lettura file*

Non è possibile rilasciare la mutua esclusione sul filesystem prima di aver completato una scrittura di un file a causa di possibili problemi di concorrenza, ad esempio il file potrebbe dover essere eliminato, oppure deve essere eliminato un altro file dall'algoritmo della cache, o ancora due istanze dell'algoritmo della cache voglio eliminare reciprocamente i file (causando deadlock).

Protocollo di comunicazione

Quando un client connesso effettua una richiesta, il tipo di richiesta viene letto da un thread worker, successivamente il thread si occupa di leggere tutti i parametri necessari ad eseguire quella richiesta nel filesystem secondo l'ordine *size, path, contenuto, flags.* A questo punto le letture sono finite e la funzione che esegue la specifica richiesta nel filesystem può essere eseguita in modo "atomico" appunto perché è indipendente dal client. Sarà la funzione del filesystem che invierà tutte e sole le risposte al client seguendo il protocollo: *0: successo, 1: invio file (size, path, contenuto), -1: fine invio file, >140 myerrno*

Scelte progettuali rilevanti

Se un client chiude la connessione mentre è in comunicazione con un thread worker, il worker manda al manager l'opposto del FD che indica che quel client deve essere chiuso, in questo modo solo il thread manager può riciclare i FD, evitando race conditions.
L'algoritmo della cache ha la precedenza su eventuali lock, cioè un file con lock può essere inviato ad un client diverso da quello che possiede la lock, questo per evitare che il server non riesca più ad accogliere file perché tutti sono lock. Un'altra soluzione ragionevole in caso i file siano di "contenuto privato" sarebbe quella di eliminare comunque il file, ma inviarlo al client solo se questo è lo stesso che ha la lock.
La openFile non riceve eventuali file eliminati perché lato client non è specificato dove salvarli.
Quando un client chiude, il filesystem viene aggiornato in modo che quel client non abbia più file aperti e tutte le lock che aveva quel client vengono rimosse.

File di log

Il path del file di log è specificato nel file di configurazione. Il file regista ogni azione interna ed esterna del server usando un linguaggio naturale.

Client

Il client è composto da un unico processo che esegue il parsing degli argomenti da riga di comando e comunica con il client tramite api. Come primo passo viene effettuato un controllo di correttezza degli argomenti e vengono settati i flag *-p* e *-t*, in seguito vengono eseguite le richieste da sinistra verso destra con la semantica che un comando non conosce mai nessun comando successivo (tranne *-t* e *-p*), quindi se si vogliono ad esempio salvare i file evict in una cartella, questa va specificata prima di specificare l'operazione di scrittura nel server che eventualmente causa evict.
Il cliente si connette al server all'avvio e si disconnette alla fine (quindi mantiene sempre aperta una sola connessione), ogni comando viene eseguito attraverso una o due richieste alle api, ad esempio per scrivere un file (opzione *-W*) c'è bisogno di una openFile e poi di una writeFile.

Rimpiazzamento

L'algoritmo di rimpiazzamento usato è un semplice FIFO ma la struttura di memorizzazione dei file permette di implementare molto facilmente anche una politica LIFO.
La funzione *cacheEvict* deve essere chiamata quando si è sicuri che è possibile recuperare spazio (in termini di spazio di memoria o file aperti), prende altri due parametri che sono il file da escludere

dall'evict (ossia quello in scrittura) e il flag che indica se è necessario eliminare un file che occupa spazio (in caso di write o append) o un file qualsiasi (in caso di open).

Gestione degli errori

La maggior parte degli errori vengono inviati dal server al client come messaggio di risposta,altri invece sono generati direttamente dall'api evitando di inoltrare una richiesta errata al server. Partono dal valore *140* per non sovrapporsi ad *errno.* In entrambi i casi, la funzione dell'api ritorna *-1* e viene settata la variabile globale *myerrno,* l'errore può essere stampato con la funzione *myperror.*

Parti opzionali svolte

- Log file del server
- Test 3
- Opzioni *lockFile* e *unlockFile*
- Invio dei file evict al client (opzione *-D*)
- Sviluppo su un repository GitHub pubblico: https://github.com/MatteoTolloso/file-storage-server