File Storage Server

1. Introduzione

File storage server è un sistema di memorizzazione di file che usa un'architettura client-server. Il client si occupa di inviare le richieste riguardo la creazione, modifica, ottenimento ed eliminazione dei file, mentre il server ha il compito di gestire il sistema di memorizzazione dei file e le varie richieste dei client.

2. Server

Il server è implementato come un singolo processo multi-threaded. A seguire è descritta la sua architettura.
Un thread manager si occupa di:

- eseguire il parsing del file di configurazione per estrapolare i parametri necessari per il corretto funzionamento del server;
- allocare e inizializzare tutte le strutture dati usate dal server, come ad esempio l'hash table, oppure le code per la comunicazione manager-worker e viceversa;
- avviare tutti i thread worker che sono previsti nel file di configurazione e che compongono il thread pool.

Al termine della configurazione del contesto di esecuzione del server, il thread manager inoltre si occupa di:

- accettare le varie richieste di connessione da parte dei client;
- eseguire il multiplexing tra le connessioni, comunicando dunque ai thread worker i descrittori dei socket su cui ricevere le richieste (per il formato dei messaggi vedere sezione 5);
- verificare lo scadere dei timeout associati alle connessioni con i client ed eventualmente chiuderle.

I thread worker si occupano di ricevere le richieste e di elaborarle, inviando infine il risultato al client.

2.1. Comunicazione tra manager e worker

Manager e worker comunicano attraverso l'inserimento e la rimozione di elementi da due code. La comunicazione tra questi è necessaria, da un lato, per informare i worker da quali socket leggere le richieste dei client, dall'altro, per informare il manager che la richiesta su un socket è stata soddisfatta.
La prima coda, chiamata 'request_queue', viene usata per la comunicazione da manager a client, segue il paradigma produttore-consumatore con un solo produttore (il thread manager) e più consumatori (i thread worker). Il manager quindi inserisce nella coda i descrittori dei socket pronti per essere letti. I thread worker rimuovono tali elementi procedendo successivamente alla gestione della richiesta.
La seconda coda, nominata 'resolved_queue', viene invece usata per la comunicazione da worker a manager, anche questa segue il paradigma produttore-consumatore, ma in questo caso abbiamo più produttori (i thread worker) e un solo consumatore (il thread manager). Questa è necessaria per informare il manager che una richiesta è stata soddisfatta su un determinato socket e quindi bisogna attendere che quest'ultimo sia nuovamente leggibile, oppure, nel caso in cui fosse necessario, procedere con la chiusura della connessione.

2.2 Memorizzazione dei file

Per quel che concerne la memorizzazione dei file, il server usa come struttura dati una hash table, dove eventuali collisioni vengono gestite inserendo i file in una linked list ordinata in ordine crescente di filename. Nel tentativo di ridurre il numero di possibili collisioni, la dimensione dell'array che modella l'hash table, viene calcolata come la dimensione massima dello storage * 1,3.

2.2.1 Funzioni hash

Un altro punto di interesse nella gestione di una hash table, risulta essere la funzione hash usata per la scelta dell'indice della cella in cui inserire un file. In questa implementazione è previsto il seguente approccio:
il sistema usufruisce di due funzioni hash uguali dal punto di vista concettuale, ma diverse sui valori dei parametri usati per il calcolo. Inizialmente si usa la prima funzione hash, se non si verifica alcuna collisione allora si procede con l'inserimento del file, in caso contrario si calcola un secondo indice con la seconda funzione hash, se anche in questo caso avviene una collisione allora si procede con il confronto tra le dimensioni delle linked list associate alle due diverse celle, selezionando come target quella con lunghezza minore per rendere più efficiente la lookup.

2.3 Replacement policy

Poiché lo storage viene gestito come un buffer con dimensione limitata, è stato necessario specificare una politica di rimpiazzamento, che nel caso di questa implementazione risulta essere LRU (Least Recently Used). Per implementare tale politica ogni file è provvisto del metadata "last_used" (per il formato dei file vedere sezione 3). Nel momento in cui è necessario rimuovere un file dallo storage l'algoritmo di

rimpiazzamento sceglie il file o i file con valore di “last_used” minore. Ad ogni operazione eseguita sul file questo valore viene aggiornato.

2.4 Identificazione dei client sul server

Per identificare i client, (necessario per comprendere quali client hanno aperto un determinato file oppure quale client possiede la lock su un file) viene usato il descrittore del socket che consente la comunicazione tra quel client e il server, questo è possibile poiché il descrittore del socket in questo caso rimane lo stesso per tutta una sessione di comunicazione tra client e server e le informazioni temporanee, come ad esempio il possesso di una lock su un file, valgono solo fino a che la sessione rimane attiva.

3. Formato dei file

Ogni file è composto da due parti: metadati e dati.
I metadati contengono informazioni relative al file stesso, tra cui abbiamo:

Campo	Descrizione	Valori
filename	Il filename associato al file. Questo è usato come identificativo univoco per il file. Contiene un filename assoluto	
size	La dimensione del file in byte	
last_used	La data in cui il file è stato usato per l’ultima volta, specificato in nanosecondi a partire da epoch	
acquired_by	Il descrittore del socket associato al client che possiede la lock sul file	-1 se la lock non è posseduta da nessuno, >0 in caso contrario
lock_type	Indica se la lock è stata richiesta con il metodo lockFile oppure con flag O_LOCK	1 se la lock sul file è stata acquisita con flag O_LOCK, 0 altrimenti
opened	L’array che contiene i descrittori dei socket associati ai client che hanno aperto il file	
index	L’indice della cella della hash table che contiene il file	
next_file	Il puntatore al file successivo nella lista, nel caso sia avvenuta una collisione nell’inserimento dei file	NULL se non ci sono state collisioni nella cella della hash table, oppure se la lista è finita
prev_file	Il puntatore al file precedente nella lista, nel caso sia avvenuta una collisione nell’inserimento dei file	NULL se il file è la testa della queue

3. Formattazione file di configurazione

Il file di configurazione è formato da una lista di coppie (nome, valore) nel seguente formato:

nome parametro:valore del parametro

una coppia per ogni riga.

I parametri che sono specificati nel file di configurazione sono i seguenti:

Codice	Descrizione
n_thread	Il numero di thread che compongono il thread pool
b_storage	La dimensione massima dello storage in Mbyte
n_file_storage	La dimensione massima dello storage in numero di file
soc_filename	Il filename del socket su cui il server accetta le nuove connessioni
max_conn_wait	Il numero massimo di connessioni in attesa di essere accettate

Codice	Descrizione
max_active_conn	Il numero massimo di connessioni attive contemporaneamente
manager_timeout	Il timeout utilizzato con la system call poll
log_filename	Il filename da usare per la creazione del file di log
client_timeout	Il timeout massimo di inattività per un client, allo scadere del tempo la connessione con il client viene interrotta, tutti i file aperti dal client vengono chiusi e tutte le lock acquisite dal client vengono rilasciate

È possibile inoltre inserire dei commenti nel file di configurazione usando il carattere '#' prima del commento. Il commento risulta essere a riga singola.

4. Formato dei messaggi scambiati tra client e server

Client e server si scambiano messaggi sotto forma di stringhe.

4.1 Formato dei massaggi di richiesta

Il messaggio di richiesta che viene inviato dal client verso il server è composto nel seguente modo:

'**Codice del comando**'["carattere separatore"**parametri aggiuntivi** …]

dove "carattere separatore" risulta essere il carattere ASCII con codice 1.
I possibili valori per il codice del comando sono descritti nella seguente tabella:

Definizione	Valore	Descrizione	Parametri aggiuntivi
CLOSECONN	0	Richiesta di chiusura della connessione	None
OPENFILE	1	Richiesta di apertura di un file	Nome del file e flag(O_CREATE, O_LOCK)
CLOSEFILE	2	Richiesta di chiusura di un file	Nome del file
WRITEFILE	3	Richiesta di scrittura di un file	Nome del file e contenuto
READFILE	4	Richiesta di lettura di un file	Nome del file
READNFILE	5	Richiesta di lettura di n file	Numero di file da leggere
APPENDFILE	6	Richiesta di scrittura in concatenazione ad un file	Nome del file e contenuto
LOCKFILE	7	Richiesta di acquisizione della lock su un file	Nome del file
UNLOCKFILE	8	Richiesta di rilascio della lock su un file	Nome del file
REMOVEFILE	9	Richiesta di rimozione di un file	Nome del file

4.2 Formato dei messaggi di risposta

Il messaggio di risposta che viene inviato dal server verso il client è composto nel seguente modo:

'**Codice di risposta**'["carattere separatore"**risultato** …]

dove "carattere separatore" risulta essere il carattere ASCII con codice 1.
I possibili valori per il codice di risposta sono descritti nella seguente tabella:

Definizione	Valore	Descrizione
SUCCESS	0	Operazione eseguita con successo
ALREADY_OPENED	1	Il file è già stato aperto
FILE_NOT_EXIST	2	Il file richiesto non esiste
UNKNOWN	3	Errore sconosciuto
FILENAME_TOO_LONG	4	Il filename richiesto è troppo lungo
FILE_ALREADY_EXIST	5	Il file richiesto esiste già
FILE_NOT_OPENED	6	Il file richiesto non è aperto
FILE_LOCKED	7	La lock sul file richiesto è posseduta da un altro utente
NOT_ENO_MEM	8	Il file richiesto ha una dimensione eccessiva

Il risultato nel messaggio di risposta dipende dall'operazione richiesta. Nel caso di operazioni di lettura, il messaggio di risposta conterrà i filename e i contenuti dei file letti, mentre nel caso di operazioni di scrittura o di apertura di un file con flag O_CREATE il messaggio di risposta potrebbe contenere i filename e i contenuti dei file espulsi dallo storage a causa del raggiungimento della soglia massima di dimensione. In tutti gli altri casi i messaggi di risposta conterranno solo il codice di risposta.

5. Formato del file di log

Il file etc/log.txt contiene al suo interno alcune informazioni riguardanti le operazioni eseguite dal server durante il suo periodo di attività. I dati sono memorizzati al suo interno nel seguente formato:

"Stringa di identificazione dell'operazione":"Informazioni sull'operazione" [timestamp]

Tra le informazioni che sono presenti nel log abbiamo:

- openlock, stringa che identifica l'apertura di un file con flag O_LOCK, è seguito dal path assoluto che identifica il file aperto;
- openfile, stringa che identifica l'apertura di un file senza flag O_LOCK, è seguito dal path assoluto che identifica il file aperto;
- writeinfo, seguito da informazioni riguardanti la scrittura di un file, come il path assoluto e la dimensione del file;
- write, riga usata per identificare la scrittura di un file, seguito dalla dimensione del file scritto, usato per semplificare il parsing del file di log durante l'esecuzione dello script statistiche.sh;
- lockfile, indica l'acquisizione della lock su un file, seguito dal path assoluto del file interessato dall'operazione;
- unlockfile, indica il rilascio della lock su un file, seguito dal path assoluto del file interessato dall'operazione;
- closefile, indica la chiusura di un file, seguito dal path assoluto del file chiuso;
- readinfo, seguito da informazioni riguardanti la lettura di un file, come il suo path assoluto e la sua dimensione;
- read, riga usata per identificare la lettura di un file, seguito dalla dimensione del file letto, usato per semplificare il parsing del file di log durante l'esecuzione dello script statistiche.sh;
- removefile, riga usata per identificare la rimozione di un file dallo storage, seguito dal path assoluto del file eliminato;
- maxsize, la dimensione massima, in byte, raggiunta dallo storage durante l'esecuzione del server;
- maxnsize, la dimensione massima, in numero di file, raggiunta dallo storage durante l'esecuzione del server;
- replacedfiles, il numero di volte che è stato rimpiazzato un file durante l'esecuzione del server;
- servedrequest, indica il numero di richieste soddisfatte dai thread worker, seguito dall'identificativo del thread e dal numero di richieste risolte dal thread, questo campo è ripetuto per ogni thread worker;

- maxactiveconn, il numero massimo di connessioni contemporaneamente attive durante l'esecuzione del server.

6. Note aggiuntive

Dopo aver eseguito i test è possibile notare che, usando lo script statistiche.sh, il numero di open con flag O_LOCK è incoerente con il numero di unlock. Questo avviene poiché nel caso si tentasse di scrivere un file già esistente, con comando -w oppure -W, il client richiederà prima l'eliminazione del file già esistente (richiedendo dunque la lock su quel file, ma successivamente poiché il file viene eliminato non viene richiesta la unlock), e solo dopo procederà con la scrittura del file. Questo comportamento è conseguenza delle specifiche delle api che richiedono, nel caso della writeFile, che l'operazione subito precedente sia la openFile(filename, O_CREATE | O_LOCK), dunque che il file sia appena stato creato. Questo comportamento si verifica ovviamente anche nel caso di una richiesta esplicita di eliminazione di un file. Infine eseguendo lo script statistiche.sh al termine dell'esecuzione del target make "test3", è possibile notare un numero elevato di richieste soddisfatte dai vari thread rispetto al numero di effettive scritture e letture riportate nel risultato. Questo accade in quanto tutti i processi client che vengono eseguiti durante il test, eseguono le stesse operazioni. Ciò comporta che la maggior parte delle operazioni non possono essere completate, in quanto le lock sui file di interesse sono già possedute da altri client e quindi non tutte le operazioni vengono riportate nel file di log.