Relazione progetto SOL

Matteo Ceccherini – matr. 607086

Introduzione

La specifica richiede la realizzazione di un file storage server che memorizza in memoria principale file che vengono inviati dai client. La capacità del server, unitamente ad altri parametri di configurazione, è definita al momento dell'avvio del server tramite un file di configurazione testuale. Il file storage server è implementato come un singolo processo multi-threaded in grado di accettare connessioni da multipli client. Ogni client mantiene una sola connessione verso il server sulla quale invia una o più richieste relative ai file memorizzati nel server, ed ottiene le risposte in accordo al protocollo di comunicazione "richiesta-risposta". Qualora un client cercasse di mandare dei file che vanno oltre alla capacità effettiva del server quest'ultimo si comporta come una cache di file implementata con una coda FIFO: il primo elemento aggiunto sarà il primo ad essere rimosso, così da far spazio al file che deve essere aggiunto.

Server

Il server consiste di un singolo processo multi-threaded, secondo lo schema "master-worker", in cui il numero di threads Worker è fissato all'avvio del server sulla base delle informazioni di configurazione (struct config_struct), ed ogni Worker può gestire richieste di client diversi. Il processo server accetta connessioni di tipo AF_UNIX da parte dei client. Vi è anche un altro thread, di supporto, per la gestione dei segnali (SignalHandler).

Tipo dati

I tipo dati utilizzati nel server sono i seguenti :

- struct config_struct : struttura per la memorizzazione dei dati passati dal file test.conf.
- Nodo_t : elemento della coda che tiene i client (identificati dal fd) che hanno aperto il file.
- Coda_t : struttura dati che tiene la testa, coda e lunghezza dei fd.
- NodoStorage_t : elemento della coda che tiene le informazione del file (nome, dati, lunghezza, flag di lock da parte di un client, fd del client che ha fatto lock, un booleano che controlla se l'ultima operazione fatta dal locker e' una open con O_CREATE_LOCK e un collegamento alla coda dei client che hanno aperto il file).
- CodaStorage_t : struttura dati che tiene le informazioni sullo storage (elemento in testa, elemento in coda, numero di file sullo storage, dimensione attuale dello storage (in bytes), numero massimo raggiunto di file memorizzati, dimensione massima (in bytes) raggiunta, numero di volte in cui l'algoritmo di cache e' stato eseguito, numero limite di file nello storage, e dimensione dello storage).
- threadArgs_t : struttura dati che viene usata per passare gli argomenti ai thread worker.
- IntConLock_t : struttura dati che tiene conto dei client connessi al server.

Segnali

La terminazione effettiva è gestita attraverso la variabile globale di stato (server_status), che assume i valori CLOSED (SIGINT, SIGQUIT) per quella immediata, o CLOSING (SIGHUP), per la chiusura più lenta con il controllo dei client attivi. Il thread main del server in caso di terminazione immediata (stato CLOSED impostato dal SignalHandler), semplicemente esce dal while di gestione delle attese. Mentre per la terminazione del server con SIGHUP chiude il socket di accettazione (di nuove connessioni) e continua a ciclare fino a quando vi sono dei client connessi. La variabile iwl, condivisa tra main e worker, contiene il numero di client attivi in ogni istante, quando diventa 0, allora il main esce dal while e procede con la chiusura.

Scelte progettuali

La struttura dati del server che realizza lo storage dei file è una coda (CodaStorage_t), tale scelta si basa sulla flessibilità, la sua intrinseca politica FIFO (adottata come politica di rimpiazzamento dei file), e sulla sua facilità di implementazione. L'elemento della coda (NodoStorage_t), contiene tra le variabili le varie info(nome,lunghezza, ecc.) ed il puntatore ad un'altra coda (Coda_t) contenente i client (identificati dal fd) che hanno aperto il file.

Client

Il programma client si occupa di effettuare il parsing delle opzioni a riga di comando, e di inviare richieste al server sulla base di esse (come richiesto nella specifica), esclusivamente attraverso l'API. Per la gestione delle opzioni si utilizza (oper_coda_t), nella quale il parsing appoggia le opzioni che poi vengono estratte. All'estrazione ogni opzione viene gestita con le opportune chiamate alle API.

Tipo dati

I tipo dati utilizzati nel client sono i seguenti :

- OperNodo_t : elemento della coda delle operazioni che tiene le informazioni sull'operazione (l'enum dell'operazione (definito in def_client.h), il flag dell'open (O_CREATE, O_CREATE_LOCK), il nome del file dell'operazione, la cartella per l'operazione (read,write,append)).
- OperCoda_t : struttura dati delle operazioni che tiene la testa, la coda e la lunghezza delle operazioni da eseguire

Scelte progettuali

Come estensione è stata sviluppata la funzione di append che permette al client di appendere ad un file presente nello storage server il contenuto di un file presente sul file sistema locale del client. Per l'invocazione sono state introdotti due nuovi argomenti che il client può utilizzare da linea di comando, nello specifico :

- -a [file, localfile] : file del server a cui fare append e file locale del file system da cui leggere il contenuto da aggiungere.
- -A [cartella] : specifica la cartella del file system del client dove memorizzare i file che il server rimuove a causa dell'aumentata dimensione del file specificato con l'opzione -a

Nella funzione openFile (API) è stato aggiunto il parametro dirname, indicante il nome della directory dove scrivere l'eventuale file espulso dal server. I flag in questione vanno inseriti con il simbolo "@" consecutivamente (senza spazi) al nome del file da aprire/creare.

Test

La cartella [src] contiene i file sorgente (divisi nelle cartelle [server] e [client]) del codice. I file di include sono all'interno della cartella [includes], mentre all'interno della cartella [Config] si trovano i file di configurazione per i test sviluppati. All'interno di [tests] si trovano gli script bash che eseguono test, mentre all'interno di [Filelog] verrà creato il file txt di filelog.. Questi ultimi file sono utili in particolare per testare l'algoritmo di rimpiazzamento di file all'interno del server che ho implementato. Invece nella cartella [Scripts] ci sono gli script bash per il test 3 (LoopClient.sh) e per le statistiche (statistiche.sh). Le cartelle [Cartella] e [Files] contengono immagini e file txt per i test così come le cartelle [Letti], [LettiFinal] e [Espulsi] sono cartelle vuote dove verranno inseriti i file in fase di test. Per eseguire il codice bisogna :

- chmod 777 Scripts/* e chmod 777 tests/*
- make test1 : compila ed esegue il test1
- make test2 : compila ed esegue il test2
- make test3 : compila ed esegue il test3
- make stat : da eseguire dopo i test per stampare le statistiche
- make clean : rimuove i file generati durante i test e gli eseguibili creati