PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI

Prof. Alessio Conte
Università di Pisa
Dipartimento di informatica
Corso triennale di informatica

Adrien KOUMGANG TEGANTCHOUANG
adrientkoumgang@gmail.com

13 luglio 2021

Indice

Capitolo 1

Presentazione generale del software

E' stato richiesto di sviluppare un file storage server in cui la memorizzazione dei file avviene in memoria principale.
Per realizzare questo progetto, ho suddiviso il lavoro principalmente in 4 parte: il server, il client, l'api di comunicazione che usa il client per comunicare con il server e un interfacia di comunicazione che fa uso l'api e il server per comunicare.

1.1 Il server

1.1.1 parametri del selver

Principalmente caratterizzato e configurato da 6 caratteristiche:

- **THREAD_WORKERS** (> 0) il numero massimo di thread workers che può creare durante la sua esecuzione
- **SIZE_MEMORY** (in Bytes e > 0) lo spazio massimo raggiungibile nella database del server
- **NUMBER_OF_FILES** (> 0) il numero massimo di files scrivibile nel server (NB: ho scelto di aggiungere questo parametro per il caso in cui qualcuno setta il server ad una dimensione molto grande lo vuole limitare sul numero di files scrivibile nel server).
- **SOCKET_NAME** il nome del socket di comunicazione che crea il server per permettere ai client di collegarsi per la comunicazione
- **LOG_FILE_NAME** il nome del file di log dove sarà scritto la story degli operazioni effettuati dal server durante la sua esecuzione
- **CONCURRENT_CLIENTS** (> 0) il numero massimo di client che il server può gestire in parallelo

1.1.2 struttura del server

Il server per la memorizzazione di ogni file, usa una struttura dati detta tabella hash (cf. my_hash.h) in cui ci memorizzo tutti i file mandati dai client. Quelli file sono memorizzati sotto forma di struttura file (cf. my_file.h) in cui abbiamo gli informazioni sul nome del file (qua il suo pathname assoluto), la dimensione della chiave, il suo contenuto, la dimensione del contenuto, l'insieme dei client che l'hanno aperto e il client che l'ha logato (se logato). La dimensione di un file è considerata come quella del suo pathname + quella del suo contenuto (tralasciando lo spazio per la memorizzazione della dimensione del pathname, del contenuto e del log).

1.1.3 funzionamento del server

Al suo avvio, il server procede alla sua configurazione, dopo aver verificato che tutto sia ben configurato si mette in attesa di richieste di comunicazioni/di elaborazione di file/messaggi da parte dei workers. Nel rispetto del numero di client che si possono connettere in parallelo, il server accetta la richiesta se non abbiamo ancora raggiunto il limite e ignora la richiesta nel caso contrario. Per ogni richiesta di elaborazione di files, il master inserisce l'identificatore del richiedente dentro un buffer di richiesta che saà ricuperato da un worker appena libero e esso si occuperà di gestire la richiesta del cliente. Il worker, appena finisce di

elaborare la richiesta del client, scriver al master dicendolo di aver finito, e secondo se era una richiesta di disconnessione oppure se la gestione della richiesta è andata male, il master disconette quel client al server.

Il server finisce la sua esecuzione in seguito alla ricezione dei segnali SIGINT, SIGQUIT (che provoca una chiusura veloce del server) e SIGHUP (che fa terminare tutte le richieste di elaborazione).

1.2 Il client

1.2.1 parametri del client

Il client dal terminale durante il suo avvio, prende in parametro i comandi da eseguire durante la sua esecuzione.

1.2.2 funzionamento del client

Il client dopo avere preso dalla riga di comando tutti i comandi da eseguire, fa un parsing su tutti quanti per verificare la validità di essi. Dopo averlo fatto, esegue i comandi detti "just once", cioè eseguiti una sola volta durante tutte l'esecuzione del client. Dopo secondo se si c'è il comando 'w' o '-W' (-r o -R) verifico la presenza del comando '-D' ('-d') che dopo conferma, lo eseguo indipendentemente se era scritto prima o dopo i comandi '-w' e '-W'.
Dopo aver fatto questi preliminari, eseguo in sequenza i comandi passati da riga di comando all'avvio del client.
dopo l'esecuzione di tutti questi comandi, il client chiama automaticamente la funzione dell'API "closeConnection".

NB: se il client ha comandi non validi, sono stampati seguito di un messaggio di help per spiegare i comandi considerati validi e il loro utilizzo.

1.3 l'API

Implementato come richiesto, contiene la possibilità di stampare degli informazioni su ogni operazione effettuato (PRINT_INFORMATION) e il motivo di un esito negativo (PRINT_REASON).

1.4 l'interfacia di comunicazione : communication

definisce gli operazioni di scrittura e di lettura che usa il server e il client per comunicare.

Capitolo 2

Autres

2.1 github

Durante tutto la sua elaborazione, il software è stato continualmente pubblicato e aggiornato sul mio github personale all'indirzzo : https://github.com/adrienKoumgangT/Project-SOL-UNIPI