Progetto di Laboratorio di Sistemi Operativi

Simone Baccile
a.a. 2018/2019

Introduzione:

Questo progetto è la realizzazione dell'Object Store. La cartella include:

- "Makefile" del progetto.
- "server_objstr.c" che implementa il lato server del progetto.
- "client_objstr.c" che implementa il client di test.
- "testsum.sh" lo script per analizzare i risultati dei test.
- La cartella "include" che contiene gli header file del progetto.
- La cartella "lib" che contiene le librerie statiche del progetto.

Per eseguire l'Object Store lanciare da terminale il comando "make all" che genererà le librerie e gli eseguibili del server e del client. In seguito lanciare il server con il comando "./server_objstr &". Per eseguire i test lanciare il comando "make test". Per l'analisi dei risultati eseguire lo script con "./testsum.sh". Con il comando "make clean" invece è possibile ripulire la directory di lavoro da tutti i file generati.

Server_objstr:

Questo file implementa il lato server del progetto. Il server implementa una gestione dei segnali SIGINT, SIGTERM, SIGQUIT in modo da permettere la corretta terminazione. Il segnale SIGPIPE viene ignorato e il segnale SIGUSR1 viene utilizzato per la stampa delle statistiche attuali del server, ma non porta alla sua terminazione. Il server inizializza la cartella "data" nella directory di esecuzione tramite la funzione "init_dir". La funzione "init_stat" invece inizializza la struttura dati che conterrà le statistiche. La funzione "init_socket" invece inizializza il file descriptor per il socket listener che sarà il socket dedicato all'ascolto di nuove connessioni. Ogni volta che il server riceve una nuova richiesta di connessione genererà un thread in modalità detached tramite la funzione "worker_spawn". Il thread generato viene inizializzato con la funzione "objstr". "objstr" legge le richieste dal socket dedicato con il client e inizializza le operazioni da effettuare. L'operazione vera e proprio viene effettuate dalle funzioni contenute in "server_op".
(N.B. All'interno di questo file e in "server_op.c" ci sono delle operazione messe come commento, poichè in una prima implementazione avevo pensato di assegnare un lock ad ogni client in modo tale che due client con lo stesso nome possono registrarsi accedendo alla stessa cartella, ma effettuando le operazioni in mutua esclusione. Queste informazioni dovevano essere inserite in una lista contenente tutti i client generati con i loro lock (le funzione di gestione della lista sono in list.h, codice non realizzato da me, ma scaricato da Github), ma per problemi sorti successivamente nella gestione della lista ho seguito le istruzioni di consegna del progetto che garantivano che i nomi dei client connessi erano tutti diversi.)

Server_op:

In questo file ci sono le implementazioni delle funzioni che deve svolgere il server. Le funzioni implementate sono:

- “ok” : questa funzione scrive sul socket di comunicazione il messaggio di operazioni eseguita con successo.
- “ko” : questa funzione scrive sul socket il messaggio di errore con la motivazione dell’errore stesso leggendo la variabile “errno”.
- “data” : questa funzione restituisce sul canale di comunicazione il dato richiesto. Viene utilizzata solamente in caso di risposta con successo dopo una richiesta di retrieve.
- “s_connect” : genera all’interno della cartella “data” una cartella con il nome con il quale il client si è connesso.
- “s_store” : si sposta all’interno della cartella del client connesso e crea un file al suo interno con un nome dato, solo se un altro file con lo stesso non esiste già. Il file verrà riempito con un blocco dati inviato dal client.
- “s_retrieve” : trova un file all’interno della cartella del client connesso e se esiste ne restituisce il contenuto al client stesso, tramite la funzione “data”.
- “s_delete : trova un file all’interno della cartella del client connesso e se esiste lo elimina.
- “s_leave” : chiama la funzione “ok” per comunicare che il client si può disconnettere.

Client_objstr:

E’ il file d’implementazione del client di test. Si appoggia sulla libreria “obj_str” per comunicare con il server. Per essere mandato in esecuzione basta eseguire “./ client_objstr nome_client n_test” dove n_test è un numero che va da 1 a 3 e serve per specificare la batteria di test da eseguire. Con la batteria di test 1 vengono salvati nello store dati di grandezza crescente. I dati generati sono definiti da #define N 20, mentre il contenuto di ogni file è dato da #define str “abcd”. Queste due direttive possono essere modificate a piacere. I dati salvati conterranno una stringa che contiene str ripetuta per un multiplo di 400. Con la batteria di test 2 vengono richiesti i tutti dati salvati nello store e vengono confrontati per verificare che corrispondano. Con la batteria di test 3 vengono eliminati tutti i dati di un cliente dalla sua cartella. Le richieste vengono effettuate da chiamate di funzioni contenute in “obj_str”.

Obj_str:

E’ il file che contiene le chiamate per comunicare con il server.

- “os_connect” : associa al client corrente un file descriptor corrispondente al socket utilizzato per la comunicazione col server. Una volta creato il socket tenta di connettersi al server con un massimo di 5 tentativi. Una volta connesso manderà il messaggio di register al server e successivamente leggerà la risposta ricevuta.
- “os_store” : manda un messaggio di store al server con i relativi dati e leggerà dal socket la risposta del server.
- “os_retrieve” : manda un messaggio di retrieve al server e leggerà la risposta. La risposta con i dati potrebbe essere spezzettata all’interno del socket, per questo in un ciclo while si continuerà a leggere dal socket fino alla fine del messaggio.
- “os_delete” : manda un messaggio di delete al server e legge la risposta.
- “os_disconnect” : manda un messaggio di leave al server ed legge la risposta.

My_utils:

Contiene alcune funzioni utili per verificare i valori di ritorno delle funzioni.