Relazione progetto LSO

Introduzione

Il progetto è diviso in tre moduli: modulo client, modulo server e modulo client test. Il modulo client viene compilato come libreria statica che verrà sfruttata dal modulo client test per eseguire dei test sul corretto funzionamento del sistema client-server.

Server

Il server gestisce le connessioni entranti dei server e l'object storage. Appena avviato carica delle informazioni essenziali di startup e si mette in ascolto sul socket objstore.sock (se il socket esiste già, il server lo elimina e ne crea un altro).

Connessioni entranti

All'arrivo di una connessione da parte di un client, il server crea un thread a lui dedicato che si occuperà di gestire le richieste del client. Il nome dei client è garantito essere unico; se un client prova a connettersi con il nome di un altro client connesso il server manda una risposta negativa. Per far ciò, il server tiene traccia di tutti i nomi dei client connessi con una hash table (implementata da hash_table.c e hash_table.h), la quale permette tempi di lookup rapidi.

Object store

Le operazioni di read/write su files vengono gestite dall'unità object store (implementata da object_store.c e object_store.h). L'object store tiene traccia del suo status (numero file presenti, numero cartelle presenti, size totale) man mano che arrivato richieste di write o delete. Lo status viene poi salvato su un file status nella cartella data alla chiusura del server e caricato al suo avvio in modo da garantire la permanenza e coerenza dei dati. I dati di un client vengono salvati in una cartella a lui dedicata dentro la cartella data con nome cartella = nome client. La scrittura dei dati avviene a blocchi (o oggetti). Ad ogni oggetto viene dedicato un file con il suo nome. I primi 8 byte dei file indicano la grandezza, in byte, dei dati contenuti in esso, in modo da velocizzare e semplificare le operazioni di retrieve.

Gestione dei messaggi

I messaggi in ingresso da parte dei client vengono gestiti dal gestore dei messaggi (implementato da message_processer.c e message_processer.h). Il gestore legge il messaggio in ingresso, traducendo il messaggio testuale in commando codificato dalla struttura command. La struttura è composta da un campo type che codifica il tipo di operazione, un campo name che codifica il primo parametro (l'uso specifico cambia a seconda del commando), un campo data contenente i dati in ingresso in caso di STORE o NULL in altri casi e un campo data_length contenente la lunghezza in byte dei dati in caso di STORE o 0 in altri casi. L'output del gestore viene poi processato dal server che provvederà a rispondere adeguatamente al client.

Segnali

Il server sfrutta un handler personalizzato per la gestione dei segnali. In caso di segnali di interrupt il server provvederà a salvare il suo stato prima di uscire. In caso di segnale SIGUSR1 il server provvederà a stampare il suo stato sullo stdout prima di terminare la sua esecuzione.

Client

La libreria client si occupa di implementare le funzioni utili alla comunicazione con l'object storage

Gestione dei messaggi

Il client usa una gestione dei messaggi simile a quella del server. Il gestore dei messaggi legge il messaggio testuale in input e lo converte in commando codificato dalla struttura command. La struttura è simile a quella del server con l'eccezione dei tipi disponibili di commando (solo OK e KO) e del campo nome, il quale qui è salvato come campo messaggio, solitamente contenente il messaggio di errore da parte del server in caso di KO.

Utils

L'header utils contiene macro, dichiarazioni ed inclusioni utili al funzionamento di client e server. In particolare contiene la macro di CHECK, il cui scopo è controllare il corretto completamento delle system call e stampare il relativo messaggio di errore in caso contrario e la macro LOG, il cui scopo è mettere a disposizione uno strumento semplice per la visualizzazione dei messaggi su console. Essa categorizza i messaggi in 4 categorie:

- INFO – messaggi semplici, solitamente messaggi di informazione sullo status del programma. Vengono stampati in verde.
- WARNING – messaggi di media urgenza, solitamente messaggi legati ad un malfunzionamento o a qualche errore non fatale durante l'esecuzione. Vengono stampati in giallo.
- ERROR – messaggi urgenti, solitamente messaggi legati ad un errore fatale durante l'esecuzione che causa la terminazione del programma. Vengono stampati in rosso.
- MESSAGGI – messaggi normali. Vengono stampati in bianco.

Test Client

Come specificato da consegna, il test client sfrutta la libreria del client per mettere a disposizione 3 tipi di test. I salvati dai test sono numerati e assumono nomi da Blocco_0 a Blocco_49. I blocchi sono di dimensione crescente, partono da 100 byte e incrementano di 5000 byte ogni volta. All'interno vengono salvate successioni di numeri da 0 a 255 fino a raggiungere la dimensione desiderata. Alla fine dei test il test client stampa il numero di test effettuati, numero di test terminati con successo e numero di test terminati senza successo in un formato tale da semplificare il controllo tramite script.

Makefile

Il makefile contiene i target per la compilazione e linking dei diversi moduli, la creazione della libreria client, il target di pulizia che rimuove tutti i file di output e il target di test che esegue lo script test.sh, il quale lancia diverse istanze del test client come specificato nella consegna.

Test Script

Il test script prende l'output dei test eseguiti dal makefile e stampa su stdout un sommario dei risultati (numero test eseguiti di tipo i, successi del tipo i, insuccessi del tipo i)

Alessio Loddo, 560008, Corso A