Object store

Progetto di Sistemi Operativi

Matteo Simone
Corso B
Matricola 560445

A.A. 2018/2019 - Secondo semestre

1 Overview

L'object store è un sistema client-server destinato alla memorizzazione di blocchi dati inviati dai client in un server.
Client e server comunicano su named socket secondo il protocollo fornito nel testo del progetto ed i blocchi vengono salvati dal server in locale.

2 Componenti

Di seguito sono illustrate le parti di cui è composto il sistema.

2.1 Server

Il server gestisce le richieste dei client tramite thread worker. In seguito ad una accept viene creato un thread worker al quale viene passato il socket corrispondente al client appena accettato e prosegue il suo ciclo di vita separato dal thread principale.

All'avvio del server vengono mascherati alcuni segnali e viene lanciato il signal handler thread per gestirli (in particolare per la gestione di SIGINT, SIGTERM e SIGUSR1). Il server si mette poi in ascolto e in attesa di connessioni tramite select; in caso di connessione riuscita il socket viene allocato nello heap tramite calloc per ovviare ad una situazione di race condition scoperta grazie all'uso di strace (a due thread diversi veniva dato lo stesso file descriptor per il socket nonostante accept ne avesse generati due diversi,

essendo però passati per riferimento il secondo veniva riutilizzato alterando il valore utilizzato in altri thread). L'id dei thread worker generati viene inserito in una lista per permettere al thread principale di aspettare il loro termine in caso di chiusura del server.

Il thread worker legge sul socket (ricevuto come argomento) tramite read. Una volta letti dei dati, tokenizza l'input per cercare un eventuale comando:

- REGISTER: viene estratto il token relativo al nome del client e viene salvato. Viene poi invocata *register_fun* che crea la cartella utente, se non esiste, e crea il path relativo ad essa.
- STORE: viene estratto il token relativo al nome del file e ne viene composto il path. Viene estratta la lunghezza e vengono letti i dati che vengono salvati in un buffer da *store_fun*, la quale scrive poi il buffer su file.
- RETRIEVE: viene estratto il token relativo al nome del file e ne viene composto il path. *retrieve_fun* apre il file precedentemente salvato, se esiste, ne calcola la lunghezza e lo scrive in un buffer inviato poi con una *write* assieme all'header.
- DELETE: viene estratto il token relativo al nome del file e ne viene composto il path. Se esiste, *delete_fun* lo elimina.
- LEAVE: viene estratto il token relativo al nome del client. *disconnect_fun* chiude il socket ed il thread termina.

All'uscita dalla sua funzione, il thread worker libera le strutture allocate dinamicamente: *usrname*, *usrpath* e *sockarg*.

Il server stampa su stderr informazioni sull'esito delle operazioni ed alcuni suoi parametri, utilizzati per la verifica del loro corretto funzionamento, e su stdout le statistiche richieste con un eventuale SIGUSR1.

2.2 Client

Il client riceve come argomento da linea di comando il nome con cui registrarsi ed un intero (compreso tra 1 e 3) per il tipo di test da effettuare, come da specifiche. Per il suo utilizzo sono stati preventivamente creati 20 file di test, alcuni dei quali contenenti stringhe di testo, altri contenenti interi progressivi ottenuti tramite seq e dd. Con seq è stato possibile ottenere un output conosciuto, ripetibile e facilmente verificabile e dd è stato utilizzato per ottenere

file di dimensione precisa (fino a 100000 byte, come da specifica). Tutti i test cominciano con una *os_connect* con il nome fornito da linea di comando e lanciano le operazioni sui file in un ciclo for con il quale viene creato il path.

- Test di tipo 1: dopo aver ottenuto il path, apre il file relativo all'iterazione, lo copia in un buffer e lo passa a *os_store*. Questa compone il messaggio (header + dati) e lo invia sul socket. Ciò viene ripetuto per il numero di file nella batteria di test.
- Test di tipo 2: dopo aver ottenuto il path, apre il file relativo all'iterazione e lo copia in un buffer. Viene poi chiamata *retrieve_fun* che invia la richiesta di RETRIEVE al server. Ricevuta una risposta verifica che l'header contenga *DATA* e in caso estrae la lunghezza. Legge poi il blocco dati, salvandolo in un buffer, e lo restituisce. Viene poi confrontato il blocco così ottenuto con il contenuto del file tramite *strncmp* per verificare la correttezza dell'operazione.
- Test di tipo 3: dopo aver ottenuto il nome del file chiama *os_delete* che invia al server la richiesta di cancellazione del suddetto.

All'uscita dal test viene stampata su stdout una stringa contenente il nome del client, il numero di operazioni lanciate e quante di queste sono state eseguite con successo. Viene poi lanciata *os_disconnect*.

2.3 Librerie

Le funzioni della Libreria di accesso sono definite in *lib/os_access.c*. Questo ha il suo header *os_access.h* ed in compilazione viene creata la libreria *libos_access.a*, che viene poi linkata staticamente al client.

In *lib/* sono presenti anche i file per la creazione di una libreria di supporto al server contenente un'implementazione minimale della linked list.

2.4 Makefile e script

Il Makefile contiene i target richiesti nelle specifiche e in aggiunta il target *clean_data* per ripulire la directory in cui vengono salvati i blocchi oggetto ricevuti dal server tramite STORE ed il target *serversum* per stampare un sommario del log del server.

Il target *test* lancia il server in background reindirizzando stderr su *server.log* e lancia *test.sh*. Viene poi lanciato *testsum.sh* ed in seguito viene chiuso il server con l'invio di un segnale.
test.sh apre 50 istanze del client, ciascuna con un nome diverso, con il test 1. Alla terminazione dei client segue il lancio di 30 istanze del client con test 2 e 20 istanze con test 3. L'output dei client viene sempre reindirizzato su *testout.log*.
testsum.sh calcola gli esiti cumulati dei test con una semplice esecuzione di awk, dopodiché invia al server SIGUSR1.
Il target *serversum*, tramite awk, stampa un breve sommario di *server.log* indicando il numero di connessioni totali, il numero di operazioni eseguite e quelle eseguite con successo ed il numero di client diversi che si sono registrati.