Object Store

Giovanni Solimeno

4 luglio 2019

Indice

1 Struttura del progetto

1.1 Struttura del codice

Tutto il codice è contenuto nella cartella `src`, dove:

- I file `commons.c` e `commons.h` contengono funzioni utili sia all'object store, che alla libreria del client, più definizioni di macro e costanti;
- I file `server.c, worker.c, signal.c` e `commands.c`, con relativi header, contengono l'implementazione dell'Object Store;
- I file `libobjstore.c` e `libobjstore.h` contengono l'implementazione della liberia lato client;
- I file `testclient.c, badclient.c` e `simtest.c` contengono dei client che testano le varie funzionalità del server. In particolare, `testclient.c` testa le funzionalità dell'Object Store, mentre i restanti test stressano l'Object Store in vari modi.

1.2 Makefile

Il `Makefile` contiene cinque target:

- Il target `all` (di default) compila l'object store, la libreria client e il client di test;
- Il target `test` compila il client di test ed esegue dei test atti a verificare il corretto funzionamento dell'object store. L'esito dei test viene ridiretto nel file `testout.log`. Si presuppone che ci sia già un Object Store in esecuzione;
- Il target `server` compila solo l'object store;
- Il target `libobjstore` compila la libreria client in formato statico, salvandola nella cartella `libs`;
- Il target `clean` rimuove tutti i prodotti del processo di compilazione.

1.3 Script di analisi

Infine, lo script `testsum.sh` elabora il file prodotto dai test, rendendolo più leggibile. Un esempio di tale processo è:

```
Format: Total   Type1   Type2   Type3
Passed: 90 40 30 20
Failed: 0 0 0 0
by:  client1 client4 client6 client5 client21 client32 client12
client25 client3 client14
```

dove i client nella riga che inizia con `by:` sono i client che hanno fallito i loro test.

Nella versione dell'Object Store allegata a questo documento, tutti i cento test dovrebbero essere eseguiti correttamente.

2 Scelte implementative

2.1 Object Store

Si è deciso di aprire il file di socket nel percorso `/tmp/objstore.sock`, in modo da rendere l'Object Store raggiungibile da qualunque client, indipendentemente dal percorso di lavoro dell'OS.
Come punto centrale del server c'è la seguente struct, definita nel file `server.h`:

```
struct server_info_s {
        ssize_t active_clients;
        int server_running;
        int server_fd;
};
```

Il membro `active_clients` mantiene il numero di clienti attivi nel server, il membro `server_running` è un flag globale che indica ad i vari thread se continuare a lavorare o no, infine, il membro `server_fd` contiene il file descriptor associato al socket del server.

Si è deciso di impedire a due client diversi di registrarsi con lo stesso nome. Le informazioni dei singoli client sono salvate in una lista formata dalle seguente struct, definita nel file `worker.h`:

```
struct worker_s {
        struct worker_s *next;
        pthread_t worker_thread;
        int worker_fd;
        int is_active;
        int is_registered;
        char associated_name[MAX_CLIENT_NAME_LEN + 1];
        struct worker_s *prev;
};
```

Il membro `associated_name` contiene il nome associato al thread del client. Ad ogni `REGISTER` ricevuta dal server, si controlla che il nome inviato nell'header non corrisponda al nome di uno dei worker nella lista dei worker attivi, in tal caso la registrazione viene rifiutata. Il membro `is_active` viene usato all'interno del `worker_loop()` per decidere se continuare a ricevere richieste dal client: Viene settato a 0 quando il client esegue una `os_disconnect`. Il membro `is_registered` serve a controllare se il client ha effettuato una `os_register` o no: Se ha valore 0, il client non può eseguire alcuna operazione all'infuori della `os_register`, che quindi aggiornerà il valore del membro a 1.

2.2 Libreria lato client

La libreria lato client espone cinque funzioni:

- `os_connect(char* nome)`, che registra il client nell'Object Store con il `nome` richiesto, va chiamata prima di tutte le altre funzioni, ed è possibile chiamarla solo una volta durante tutto il periodo di utilizzo dell'Object Store dal parte del client;
- `os_store(char* nome, void* data, size_t len)`, che salva all'interno dell'object store i `len` bytes puntati da `data` chiamati `nome`;
- `os_retrieve(char* nome)`, che recupera dall'Object Store i dati identificati dal `nome`;
- `os_delete(char* nome)`, che rimuove dall'Object Store l'elemento chiamato `nome`;
- `os_disconnect()`, che disconnette il client dall'Object Store.

3 Gestione dei segnali

L'Object Store gestisce tre segnali, `SIGINT, SIGTERM` e `SIGUSR1`, dei quali solo i primi due terminano il programma:

- Sia `SIGINT` che SIGTERM sono gestiti allo stesso modo: Il server inizia le procedure di spegnimento, chiudendo il file descriptor del socket. L'Object Store imposta il valore del membro `server.server_running` a 0, comunicando ai thread attualmente attivi di terminare. L'Object Store attende dunque che tutti i thread terminino, prima di uscire dal main.
- Quando l'Object Store riceve il segnale `SIGUSR1`, stampa il numero di client connessi, il numero di file salvati all'interno dell'Object Store e il peso totale dei dati salvati. Il programma prosegue dunque l'esecuzione.

4 Esecuzione dei test

Eseguire il comando `make`. In un'altra shell, eseguire il server (file `server.o`), e in quella principale eseguire `make test`