Università di Pisa
Dipartimento di Informatica
Corso di Laurea in Informatica

Progetto di Laboratorio di Sistemi Operativi

a.a. 2018-19

Introduzione

Lo studente dovrà realizzare un *object store* implementato come sistema client-server, e destinato a supportare le richieste di memorizzare e recuperare blocchi di dati da parte di un gran numero di applicazioni. La connessione fra clienti e object store avviene attraverso socket su dominio locale.
In particolare, sarà necessario implementare la parte server (object store) come eseguibile autonomo; una libreria destinata a essere incorporata nei client che si interfacci con l'object store usando il protocollo definito sotto; e infine un client di esempio che usi la libreria per testare il funzionamento del sistema.

L'object store

L'object store è un eseguibile il cui scopo è quello di ricevere dai client delle richieste di memorizzare, recuperare, cancellare blocchi di dati dotati di nome, detti "oggetti". L'object store gestisce uno spazio di memorizzazione separato per ogni cliente; i nomi degli oggetti sono garantiti essere univoci all'interno dello spazio di memorizzazione di un cliente, e i nomi dei clienti sono garantiti essere tutti distinti. Tutti i nomi rispettano il formato dei nomi di file POSIX.
L'object store è un server che attende il collegamento di un client su una socket (locale) di nome noto, `objstore.sock`. Per collegarsi all'object store, un client invia al server un messaggio di *registrazione* nel formato indicato sotto; in risposta, l'object store crea un thread destinato a servire le richieste di quel particolare cliente. Il thread servente termina quando il client invia un esplicito comando di *deregistrazione* oppure se si verifica un errore nella comunicazione. Le altre richieste che possono essere inviate riguardano lo *store* di un blocco di dati, il *retrieve* di un blocco di dati, e il *delete* di un blocco di dati. I dettagli del protocollo sono dati nel seguito.

Internamente, l'object store memorizza gli oggetti che gli vengono affidati (e altri eventuali dati che si rendessero necessari) nel file system, all'interno di file che hanno per nome il nome dell'oggetto. Questi file sono poi contenuti in directory che hanno per nome il nome del client a cui l'oggetto appartiene. Tutte le directory dei client sono contenute in una directory `data` all'interno della working directory del server dell'object store.

Il server quando riceve un segnale termina il prima possibile lasciando l'object store in uno stato consistente, cioè non vengono mai lasciati nel file system oggetti parziali. Quando il server riceve il segnale SIGUSR1, vengono stampate sullo standard output alcune informazioni di stato del server; tra queste, almeno le seguenti: numero di client connessi, numero di oggetti nello store, size totale dello store.

La libreria di accesso

La libreria lato client che fornisce l'accesso all'object store deve fornire all'applicazione cliente le seguenti funzioni:

- **int os_connect(char *name)** – inizia la connessione all'object store, registrando il cliente con il *name* dato. Restituisce *true* se la connessione ha avuto successo, *false* altrimenti. Notate che la connessione all'object store è globale per il client.
- **int os_store(char *name, void *block, size_t len)** – richiede all'object store la memorizzazione dell'oggetto puntato da *block*, per una lunghezza *len*, con il nome *name*. Restituisce *true* se la memorizzazione ha avuto successo, *false* altrimenti.
- **void *os_retrieve(char *name)** – recupera dall'object store l'oggetto precedentemente

memorizzatato sotto il nome *name*. Se il recupero ha avuto successo, restituisce un puntatore a un blocco di memoria, allocato dalla funzione, contenente i dati precedentemente memorizzati. In caso di errore, restituisce NULL.

- **int os_delete(char *name)** – cancella l'oggetto di nome *name* precedentemente memorizzato. Restituisce *true* se la cancellazione ha avuto successo, *false* altrimenti.
- **int os_disconnect()** – chiude la connessione all'object store. Restituisce *true* se la disconnessione ha avuto successo, *false* in caso contrario.

Tutte le funzioni della libreria devono dialogare con il server tramite il protocollo definito sotto. Gli eventuali errori di comunicazione vanno gestiti e segnalati al chiamante tramite valori di ritorno come indicato sopra.

Protocollo di comunicazione

Tutti i messaggi scambiati fra clienti e object store hanno un formato basato su una riga di testo (*header*) codificato in ASCII, terminata da un carattere newline (“\n”); la riga di header può essere in alcuni casi seguita da un numero prefissato di dati binari. La lunghezza di questi dati è espressa, in ASCII, nella riga di header.

Tutte le comunicazioni sono iniziate dal client (*request*), e prevedono una risposta da parte dell'object store *(response*). Il response può contenere un *message* in testo libero, a discrezione dello studente, che specifica la causa degli eventuali errori.

In dettaglio, il protocollo prevede i seguenti messaggi:

- Registrazione: inviato dal cliente per registrarsi sull'object store; quest'ultimo risponde con l'esito dell'operazione
  - Request: **REGISTER *name* \n**
  - Response: **OK \n** (registrazione riuscita) oppure **KO *message* \n** (registrazione fallita)
- Memorizzazione: inviato dal cliente per memorizzare un oggetto nel proprio spazio privato sull'object store; quest'ultimo risponde con l'esito dell'operazione.
  - Request: **STORE *name len* \n *data*** (in cui *data* è un blocco binario di lunghezza *len* bytes)
  - Response: **OK \n** (memorizzazione riuscita) oppure **KO *message* \n** (memorizzazione fallita)
- Lettura: inviato dal cliente per recuperare i dati di un oggetto precedentemente memorizzato nel proprio spazio privato sull'object store; quest'ultimo risponde con i dati richiesti, oppure una segnalazione di errore.
  - Request: **RETRIEVE *name* \n**
  - Response: **DATA *len* \n *data*** (se l'operazione ha avuto successo, in cui *data* è un blocco binario di lunghezza *len* bytes) oppure **KO *message* \n** (in caso di fallimento)
- Cancellazione: inviato dal client per cancellare un oggetto memorizzato sull'object store; quest'ultimo risponde con una indicazione di successo.
  - Request: **DELETE *name* \n**
  - Response: **OK \n** (cancellazione riuscita) oppure **KO *message* \n** (cancellazione fallita)
- Disconnessione: inviato dal client per chiudere in maniera ordinata la connessione verso l'object store; ha sempre successo.
  - Request: **LEAVE \n**
  - Response: **OK \n**

Test

Il client di test deve esercitare tutte le funzioni offerte dalla libreria, con cui dovrà essere linkato staticamente. A tale scopo, dovrà essere in grado, in esecuzioni distinte, di

1. creare e memorizzare oggetti. Il client dovrà generare 20 oggetti, di dimensioni crescenti da 100 byte a 100000 byte, memorizzandoli sull'object store con nomi convenzionali. Gli oggetti dovranno contenere dati "artificiali" facilmente verificabili (per esempio: byte di valore numerico consecutivo 0, 1, 2... oppure sequenze di 8 byte di valore 8, poi 9 byte di valore 9, 10 byte di valore 10 ecc., oppure più copie di una stringa di testo prefissata, ecc.)
2. di recuperare oggetti verificando che i contenuti siano corretti
3. di cancellare oggetti

Il client riceverà come argomento sulla riga di comando il nome cliente da utilizzare nella connessione con l'object store, e un numero nell'intervallo 1-3 corrispondente alla batteria di test da effettuare (come indicato sopra). Terminati i test della batteria richiesta, il client deve uscire, e stampare sul suo *stdout* un breve rapporto sull'esito dei test (numero di operazioni effettuate, numero di operazioni concluse con successo, numero di operazioni fallite, ecc.). Il formato di questo report è a discrezione dello studente.

Makefile

Il progetto dovrà includere un makefile avente, fra gli altri, i target *all* (per generare l'eseguibile dell'object store, la libreria, e l'eseguibile del client di test), *clean* (per ripulire la directory di lavoro dai file generati), e *test.* Quest'ultimo deve eseguire un test dell'object store, lanciando dapprima in contemporanea 50 istanze del client di test (ciascuna con nome diverso), che effettueranno test di tipo 1. Terminata l'esecuzione di queste istanze, vanno lanciate – sempre in contemporanea – altre 50 istanze (con gli stessi nomi usati in precedenza), di cui 30 devono eseguire test di tipo 2, e 20 test di tipo 3. L'output cumulato dei test, con eventualmente altre informazioni utili, deve essere memorizzato in un solo file di nome `testout.log` nella directory corrente.

Script di analisi

Lo studente dovrà realizzare un semplice script bash di nome `testsum.sh` che legga il contenuto di `testout.log`, e calcoli e stampi su *stdout* un sommario dell'esito dei test (clienti lanciati, clienti che hanno riportato anomalie, numero di anomalie per batteria di test, ecc.). Il tipo di sommario e il formato esatto è a discrezione dello studente. Si cerchi comunque di privilegiare il tipo di informazioni che possono essere utili durante lo sviluppo per accertarsi che tutto il sistema object store+client funzioni correttamente.

Infine, lo script manda un segnale SIGUSR1 al server.

Note finali

Ci si attende che tutto il codice sviluppato sia, per quanto possibile, conforme POSIX. Eventuali eccezioni vanno documentate nella relazione di accompagnamento.
La consegna dovrà avvenire, entro il termine pubblicato, attraverso l'upload sul sito Moodle del corso di un archivio con nome *nome_cognome*.tar.gz contenente tutto il necessario. Scompattando l'archivio in una directory vuota, dovrà essere possibile eseguire i comandi *make* (con target di default) per costruire tutti gli eseguibili, e *make test* per eseguire il test e vederne i risultati. L'archivio dovrà anche contenere una breve relazione (3-5 pagine) in formato PDF in cui illustrate il vostro progetto.