RELAZIONE PROGETTO SISTEMI OPERATIVI
a.a. 2018/2019

Matteo De Francesco
matricola: 562176

Indice:

1. Scelte progettuali

1.1 Lista di appoggio e gestione concorrenza

Come struttura dati di appoggio il server utilizza una lista linkata, in cui viene inserito un client al momento della connessione e viene rimosso al momento della chiusura del thread o in caso di disconnessione del client durante le operazioni.
Si utilizza la lista in modo che se un client prova a collegarsi con un nome già presente, questo viene subito disconnesso.
Ogni funzione della libreria *queue.h* utilizza una variabile di mutua esclusione che viene bloccata all'inizio di ogni funzione e sbloccata al termine. Ciò garantisce che ogni volta che viene chiamata una funzione sulla lista la modifica avviene in mutua esclusione.
Le funzioni di *queue.h* sono:

- *start_queue:* inizializza la lista;
- *push_queue:* chiama la *check_queue* e aggiunge un client se e solo se questo client non è già presente in lista;
- *pop_queue*: rimuove un client dalla lista;
- *destroy_queue:* per fare garbage collector della memoria allocata dalla lista;
- *check_queue*: usata dalla push_queue per controllare la lista.

1.2 Gestione Segnali

Il server utilizza un handler dei segnali SIGINT, SIGTERM, SIGQUIT, SIGTSTP e SIGUSR1 per poterli gestire diversamente da quello predefinito. Nel caso di SIGUSR1 viene settata la variabile *usr e* nel thread principale del server vengono stampate le seguenti informazioni:

- numero di client connessi al momento(*numero_attivi*);
- numero di oggetti presenti nell'objstore(*numero_oggetti_store*);
- dimensione totale dell'objstore(*size_totale_store*);

Per quest'ultimo dato viene usata una funzione di appoggio *get_size* che visita ricorsivamente la directory data e somma le dimensioni di ogni file (ottenute con fstat e *st_size)* alla variabile globale *size_totale_store*. Al termine della stampa delle info, *size_totale_store* e *usr* vengono resettate e prosegue il funzionamento del server.

2 Gestione del server

2.1 Funzionamento generale

Il main thread del server crea il socket, la cartella data dell'objstore e la maschera dei segnali e si mette in attesa sulla *accept.* Al momento dell'accettazione di una connessione viene lanciato un nuovo thread che gestisce quel client e se *usr*=1 vengono stampate le info del server. Ogni thread prende come argomento il file descriptor restituito dalla accept. A seguire viene chiamata la *pthread_detach,* in modo che alla chiusura del thread vengono liberate tutte le risorse occupate. Ogni thread *worker* che viene lanciato esegue le operazioni richieste dal client. La condizione di terminazione di un thread è la variabile *terminato* nel caso dell'arrivo di un segnale e la variabile *r* che viene settata a 0 in caso di LEAVE oppure -1 in caso di errori nelle system call. Ogni thread esegue una lettura byte per byte fino al *n* per ottenere la richiesta del client. In seguito entra nell'opportuno caso:

- REGISTER: se il client è gia presente nella lista dei connessi, il thread manda un KO al client ed entrambi terminano. Altrimenti il client si connette e la variabile globale *numero_attivi* viene incrementata. In particolare, se è la prima connessione viene creata anche la directory personale;
- Caso STORE: legge un carattere a vuoto e poi continua a leggere il dato dal client, la cui lunghezza è contenuta nel messaggio precedente. Se il salvataggio del dato è andato a buon fine manda un messaggio di OK al client, altrimenti manda un messaggio di KO;
- Caso RETRIEVE: cerca il dato richiesto dal client all'interno della sua directory personale. Se il dato viene recuperato con successo manda un messaggio contenente *DATA "lunghezza del dato"* *n "dato effettivo"*, altrimenti manda un KO;
- Caso DELETE: rimuove un dato dall'objstore. Manda un messaggio di OK al client se il dato è stato rimosso correttamente, altrimenti manda un KO;
- Caso LEAVE: manda in ogni caso un messaggio di OK al client e esegue la routine di disconnessione (rimozione del client dalla coda, decremento della variabile *numero_attivi*, chiusura del file descriptor).

In caso di errori in lettura o scrittura il risultato è 0 se il client si è disconnesso inavvertitamente, quindi viene eseguita la routine di disconnessione (rimozione del client dalla coda, decremento della variabile *numero_attivi*, chiusura del file descriptor).
Le variabili globali *numero_attivi* e *numero_oggetti_store* vengono modificate in mutua esclusione.

2.1 Libreria del server

Il thread worker utilizza una libreria *server_library.h* per eseguire le operazioni sull'objstore:

- *registra_nuovo_cliente:* viene chiamata nel caso REGISTER per creare la directory del client (passato come parametro) in caso di primo accesso. Ritorna 1 se la directory è stata creata, 0 se la directory è gia esistente;
- *salva_dato_cliente:* viene chiamata nel caso STORE e accede alla directory del client (passato come parametro) per salvare un nuovo dato all'interno dell'objstore. In caso di dato già presente, il dato vecchio viene eliminato e quello nuovo viene salvato. Ritorna 1 se il dato è stato salvato correttamente, -1 se non è stato possibile salvarlo;
- *recupera_dato_cliente:* viene chiamata nel caso di RETRIEVE, accede alla directory del client (passato come parametro) e recupera il dato richiesto. Ritorna NULL in caso di dato non esistente, altrimenti un puntatore al dato;
- *cancellazione_dato_cliente:* viene chiamata nel caso di DELETE, accede alla directory del client (passato come parametro) per eliminare un dato. Ritorna 1 se il dato è stato eliminato correttamente, 0 se il dato non è presente.

Ogni funzione della libreria ritorna -2 in caso di errori nelle system call. Il thread in questo caso manda un KO al client e continua a gestire le richieste successive.

3 Client

3.1 Funzionamento generale e libreria

Il client prende come argomenti il suo nome e la batteria di test da eseguire. Nella prima batteria vengono eseguite 20 store, nella seconda viene eseguita una retrieve, nella terza una delete. Ognuno prima di eseguire la relativa batteria di test si connette al server e alla chiusura si disconnette. Prima di terminare stampa delle informazioni sulle operazioni eseguite: numero di operazioni effettuate, numero di operazioni con successo, numero di operazioni fallite.
La libreria del client contiene un intero globale che rappresenta il socket a cui si connette. Rispetta il protocollo descritto nel testo:

- *os_connect:* connette un client tramite *socket e connect*, il client manda la richiesta e viene connesso con eventuale creazione della cartella, restituisce true(1) in caso di successo. Se fallisce, perche già connesso, viene immediatamente terminato restituendo false(0);
- *os_store:* manda la richiesta di salvataggio di un dato e ritorna true(1) in caso di successo, 0 altrimenti;
- *os_retrieve:* manda la richiesta al server, e legge la risposta con due read separate: prima una read fino al *n* di *DATA "len"* *n*, poi conoscendo *len* legge il dato di risposta. Restituisce true(1) se il recupero del dato ha avuto successo, false(0) se riceve un KO;

- *os_delete:* manda la richiesta al server. Restituisce true(1) se la cancellazione ha avuto successo, false(0) altrimenti;
- *os_disconnect:* manda la richiesta al server e legge la risposta. Restituisce true(1).

In tutte le funzioni in caso di errori in lettura/scrittura, se il risultato è -1 oppure 0 il client viene terminato immediatamente.

Appendice A: Sistemi operativi provati per il testing

Ubuntu 18.04.2 LTS
Xubuntu 14.04

Appendice B: libreria condivisa util.h

La libreria condivisa *util.h* viene usata sia dal server che dal client. Contiene la funzione *Malloc* (*malloc* con controllo degli errori) e varie *define* utili per entrambi.

Appendice C: batteria di test e testsum.sh

La batteria di test *test.sh* lancia 50 istanze di client che eseguono test di tipo 1(20 store), dopodiche lancia 50 istanze di client di cui 30 eseguono test di tipo 2(retrieve) e 20 che eseguono test di tipo 3(delete).
Lo script in bash *testsum.sh* stampa su stdout la somma di tutte le operazioni effettuate, tutte le operazioni con successo e quelle fallite. Infine se si verificano errori in alcuni test, il test stesso e il nome del client vengono stampati, se si verificano errori in connessione viene stampato solo il nome.