Relazione di
Emanuele Urselli
Matricola: 559313
Corso: B

Per la realizzazione del progetto, per ogni componente, di seguito, sono descritte le implementazioni e il motivo della scelta di queste ultime.

OBJECT STORE

L'object store rappresenta il server.
È in grado di accettare numerose richieste di connessioni da parte di client ed utilizza il metodo "un thread per connessione", ovvero, per ogni richiesta di connessione, genera un thread worker che gestirà la comunicazione e le richieste fino alla disconnessione del client.
Il server gestisce uno spazio di memorizzazione separato per ogni client, rappresentato dalla directory 'data'.
Per le comunicazioni col client, si rispetta il protocollo di comunicazione assegnato nella specifica del progetto.
L'object store fa uso di 4 variabili per accedere in mutua esclusione sia al file system che ad alcune variabili che saranno utili per il calcolo del resoconto dello stato del server, ovvero: variabile globali che memorizzano il numero di client connessi, il numero di oggetti memorizzati e la sua size totale.
I nomi dei client connessi sono memorizzati all'interno di una struttura dati implementata come una coda.
Si sarebbe potuta utilizzare una tabella hash per avere una soluzione efficiente oppure un array per una soluzione meno efficiente.
Si è voluto optare per un compromesso.
Appena avviato, il server setta i thread worker in modalità detached e setta anche la gestione dei segnali.
Alla ricezione del segnale SIGUSR1, stampa il resoconto dello stato del server, senza terminare la sua esecuzione, mentre alla ricezione dei segnali SIGQUIT o SIGTERM, termina la sua esecuzione, lasciando il tutto in uno stato quanto più consistente possibile.
Dopo aver effettuato questi compiti, si mette in attesa infinita di richieste di connessioni.

WORKER

Il thread worker, come detto sopra, gestisce tutta la comunicazione con un client, fino alla disconnessione di quest'ultimo.

Appena generato, avvia un ciclo infinito per l'ascolto delle richieste da parte del client.
Alloca dinamicamente con una dimensione prefissata, uno spazio di memoria che conterrà l'header inviato dal client e poi si mette in attesa di riceverlo.
Appena ricevuto rialloca la memoria dell'header con la sua dimensione effettiva e inizia ad estrarre dall'header il comando.
Controlla il tipo di comando e se:

- **REGISTER:** La register è la funzione fondamentale per la comunicazione col client. Deve essere sempre fatta prima di eseguire qualsiasi altra funzione. La register estrae il nome del client dall'header e lo cerca nella struttura dati che contiene i client attualmente connessi. Se ne trova uno con lo stesso nome, già connesso, allora rifiuta la connessione del nuovo client, altrimenti, lo aggiunge alla struttura dati. Se l'inserimento nella struttura dati va a buon fine, incrementa il numero dei client connessi e controlla l'esistenza di una directory col nome del client appena connesso. Se esiste, da il diritto al client di accedervi ed effettuare le operazioni descritte di sotto, altrimenti ne crea una.
  Infine, riporta l'esito finale dell'operazione al client.
- **STORE:** Estrae il nome dell'oggetto da memorizzare, estrae la lunghezza dei dati da memorizzare e infine estrae i dati. Definisce il pathname del file costituito da nomeclient/nomeoggetto (nomeclient corrisponde al nome della directory). Se il file non esiste, lo crea, altrimenti lo sovrascrive controllando che tutti i dati siano stati scritti correttamente. Al termine di queste operazioni, invia l'esito al client, liberando la memoria allocata per il pathname e chiudendo il file descriptor del file.
- **RETRIEVE:** Estrae il nome dell'oggetto, come nella store definisce un pathname costituito da nomeclient/nomeoggetto. Con una funzione creata appositamente (isThere) controlla se il file esiste. Nel caso non esista, invia al client un messaggio riportando la relativa informazione, altrimenti, accede al file e se ne calcola la lunghezza. Crea un altro header (testata) che conterrà la keyword DATA accompagnato dalla lunghezza. Una volta estrapolati i dati dal file, li concatena all'header e lo invia al client, liberando la memoria precedentemente allocata e chiudendo il file descriptor del file.
- **DELETE:** Estrae, come al solito, il nome dell'oggetto. Definisce un pathname costituito da nomedirectory/nomeoggetto. Sempre attraverso la funzione isThere, controlla l'esistenza del file. Se non esiste, avvisa subito il client, altrimenti lo rimuove. Infine riporta

l'esito dell'operazione al client, liberando la memoria allocata.

- **LEAVE:** La leave è l'ultima funzione che viene eseguita. Le operazioni definite al suo interno sono eseguite, ovviamente, anche nel caso di errori "fatali" nelle operazioni precedenti. Altrimenti, se tutte le altre operazioni vanno a buon fine, sarà compito del client chiamare la leave per eseguire una corretta disconnessione che comporterà il decremento dei client connessi al server, la chiusura della directory sulla quale il client ha effettuato determinate operazioni e l'eliminazione del nome del client dalla struttura dati.

Una volta uscito dal ciclo, il worker si accerta che sia stata eseguita una leave, attraverso il controllo del valore della variabile conn. Se la variabile conn ha ancora valore uno, vuol dire che la leave non è stata eseguita e che quindi, devono essere effettuate le operazioni definite all'interno della funzione leave.
Infine, libera tutta la memoria dinamicamente allocata e chiude il file descriptor del scoket, terminando la sua esecuzione.

Il server oltre ad utilizzare la libreria "clientconnessi" per le elaborazioni sulla struttura dati che mantiene i nomi dei client attualmente connessi all'object store, utilizza anche un header "headerserver.h" in cui sono definiti tutti i tipi di messaggi da inviare al client oppure da stampare sul proprio stdout e delle MACRO utilizzate per il controllo dei valori di ritorno delle funzioni.

LIBRERIA: accesso.a
La libreria accesso.a fornisce al client le funzioni per interfacciarsi col server.

- **OS_CONNECT:** La connect è la prima funzione che il client invoca. È la funzione che permette al client di connettersi con l'object store. Alloca memoria per l'header. Esegue le chiamate di sistema socket, e connect. Se la connect va a buon fine, copia nell'header il comando REGISTER e concatena il nome del client ricevuto come argomento. Al termine, invia l'header al server e si mette in attesa di una risposta. Alla ricezione, compara il messaggio con alcuni messaggi definiti nel file header "headerclient". Se il messaggio non corrisponde a nessun messaggio di errore, libera la memoria allocata dinamicamente precedentemente e ritorna
  un valore che indicherà al client che la connessione è stata effettuata correttamente.
- **OS_STORE:** La store prende come argomenti il nome dell'oggetto,

i dati da memorizzare e la lunghezza dei dati. Prepara un header, in cui copia il comando STORE e concatena il nome dell'oggetto da memorizzare, la lunghezza dei dati e i dati stessi.
Invia l'header al server e si mette in attesa di una risposta.
Anche nella store, il messaggio di risposta del server sarà comparato con possibili errori definiti nell'headerclient. Se non ci sarà nessun match, allora ritornerà al client un valore che indicherà la corretta memorizzazione dell'oggetto all'interno dello store.

- **OS_RETRIEVE:** La retrieve è la funzione che permette di estrarre i dati da un oggetto nello store. La retrieve, come le altre, definisce un header costituito dal nome del comando RETRIEVE e dal nome dell'oggetto dal quale estrarre i dati ritornare al client. Invia l'header al server e attende il messaggio di risposta. Il messaggio può essere un messaggio di errore oppure può contenere un header del tipo "DATA lunghezza dati \n dati". Quindi, come le altre funzioni, alla ricezione, controlla la risposta con i messaggi di errori predefiniti nel file header del client. Se non combacia con nessuno dei possibili errori, vuol dire che il messaggio ricevuto contiene i dati e quindi ritornerà al client un puntatore ad essi.
- **OS_DELETE:** La delete permette al client di eliminare un oggetto dalla directory col suo stesso nome, all'interno dello store. La delete definisce un header nel quale copierà il comando DELETE e concatenerà il nome dell'oggetto che il client vuole eliminare. Una volta terminata la composizione dell'header, lo trasmette all'object store e si mette in attesa di un messaggio da parte del server. Come al solito, alla ricezione, comparerà questo messaggio con i messaggi definiti nel file header e se non ci sarà nessun match con messaggi di errore, ritornerà un valore al client che indicherà la rimozione corretta all'interno dello store.
- **OS_DISCONNECT:** La disconnect non prende in ingresso alcun argomento e definisce un header costituito solo dal comando LEAVE. Lo invia al client e attende la risposta. In questo caso i possibili messaggi sono solo due. Disconnessione effettuata correttamente oppure no. Ritornerà al client un valore che indicherà uno dei messaggi appena descritti. Come nelle altre funzioni, è sempre Possibile avere un errore di comunicazione che viene comunque controllato. Anche questo errore, ovviamente, corrisponde ad un valore che potrebbe essere ritornato al client.

CLIENT

Il client, come da specifica, prende in ingresso il nome del client che sarà utilizzato per effettuare la connessione all'objectstore e un numero da 1 a 3 che indica la batteria di test da eseguire.
Dopo aver controllato che gli argomenti rispettino determinate restrizioni, esegue la connessione al server con la funzione connect. Se la connect fallisce, il client termina subito con un fallimento, altrimenti si proseguirà con l'esecuzione della batteria di test.
Ogni batteria di test è costituita da un ciclo di 20 iterazioni.
Nella batteria di test di tipo 1, saranno eseguite 20 operazioni di store e per ognuna è controllato il valore di ritorno e stampato un messaggio corrispondente.
Così come nella batteria di test di tipo 1, le batterie di test di tipo 2 e 3 eseguono esattamente un ciclo di 20 iterazioni, nel primo caso costituito solo da operazioni di retrieve e nel secondo caso da operazioni di delete. A differenza della connect, un errore nelle operazioni della batteria di test, non provoca il fallimento totale del client ma solo un incremento delle operazioni fallite che, insieme al numero di operazioni effettuate e quelle terminate con successo, andrà a costituire il messaggio di resoconto che verrà stampato alla fine di ogni batteria di test.
Prima di stampare il messaggio, e quindi, prima di stampare il resoconto, verrà eseguita la disconnessione.

SCRIPT
Il test del make file è eseguito da uno script, chiamato script.sh.
Questo script eseguo prima un ciclo in cui legge dal file nomi.txt 50 nomi utilizzati per lanciare client differenti che eseguono la batteria di test di tipo 1. Dopo questo primo ciclo ne esegue altri due leggendo sempre nomi da file ma eseguendo 20 iterazioni che lanciano client che eseguono batteria di test di tipo 3 e 30 iterazioni che lanciano client che eseguono la batteria di test di tipo 2.

SCRIPT DI ANALISI
Esegue un ciclo che legge riga per riga il file testout.log, prodotto alla fine del test descritto sopra. Grazie alla struttura di testout.log, controlla il numero di client lanciati, controllando il numero di stringhe "CLIENT AVVIATO" presenti nel file. Inoltre, nello stesso ciclo, conta il numero di volte in cui sono stati eseguiti i diversi tipi di test, incrementando le rispettive variabili e infine stampa un resoconto, inviando al server il comando SIGUSR1 che produrrà sul suo stdout(del server) il resoconto dello stato dell'objectstore.