Relazione progetto Sistemi Operativi e Lab.

Serafino Gabriele – Matricola 564411 - 30/07/2019

INTRODUZIONE

Le specifiche del progetto richiedono la realizzazione in linguaggio C di un Object Store, ovvero di un server multithreaded che permette agli utenti di collegarsi, memorizzare, leggere ed eliminare dati detti "oggetti" identificati da un nome.
Deve essere garantito il corretto funzionamento dell'Object Store anche nel caso vi sia un numero molto elevato di utenti connessi contemporaneamente.
Viene richiesto inoltre di fornire una libreria che permetta di eseguire le operazioni descritte ed un client che utilizzi questa libreria per testare le funzionalità del server.

SCELTE IMPLEMENTATIVE

Univocità degli username

Il server deve garantire l'univocità dei nomi utente all'interno di una sessione: due client infatti non possono essere connessi contemporaneamente all'OS se hanno lo stesso username.
Per far ciò il server mantiene in una hashtable la lista di username connessi in quel determinato momento.

- Alla connessione di un nuovo utente il server cerca nella HT l'username che gli è stato fornito
- Se l'username è già presente la richiesta di connessione fallisce
- Se l'username non è presente viene aggiunto alla HT e la richiesta di connessione viene accettata
- Alla disconnessione di un client l'username ad esso associato viene rimosso dalla HT, in modo da renderlo libero per un utilizzo futuro.

L'uso in particolare della hashtable come struttura dati permette un costo operazionale molto minore rispetto a quello di un'altra struttura come una semplice linked list, permette quindi di mantenere il server veloce in queste particolari operazioni anche in caso di moltissimi utenti collegati.

Stato del server

Un'altra richiesta del progetto è quella di gestire il segnale SIG_USR1 stampando in output un resoconto dello stato del server. L'output del server contiene:

- La lista degli utenti connessi
- Il numero e la dimensione totale di tutti gli oggetti memorizzati nell'OS
- Per ogni utente il numero e la dimensione di oggetti che egli ha memorizzato

Per evitare di dover scorrere l'intera cartella di ogni singolo utente si è preferito utilizzare ancora una volta una hashtable, la quale contiene per ogni utente (anche quelli non connessi in quel momento) un elemento di tipo "status" con l'username, il numero di oggetti e la dimensione in byte, in questo modo ad ogni operazione eseguita da un utente i suoi dati vengono aggiornati.
Quando il server riceve il segnale SIG_USR1 basta scorrere la HT e stampare i dati degli utenti.
Il contenuto della HT, alla chiusura del server, viene memorizzato sulla memoria di massa in un file chiamato "status.dat" e al successivo riavvio i dati vengono caricati nuovamente da questo file ed il server è pronto a ripartire in uno stato consistente.

Terminazione del server

La gestione di più client contemporanei viene effettuata con l'utilizzo di più thread che gestiscono le richieste. I thread non sono organizzati in un pool, ma ad ogni connessione di un nuovo client viene effettuato lo spawn di un thread in modalità detatched, e alla disconnessione il thread viene terminato.

In questo modo non è necessario tenere traccia degli handle ad ogni singolo thread per eseguire poi la join su ognuno di essi.

Quando il thread main del server riceve un segnale di terminazione setta un flag globale. Ogni thread quindi esegue la read (chiamata bloccante) all'interno di una select con timer, in modo da controllare costantemente il flag di terminazione. Perciò ogni thread si accorge del flag e termina il prima possibile concludendo l'eventuale operazione in cui era impegnato.

Inoltre ogni thread prima di terminare decrementa una variabile globale che tiene conto del numero di thread attivi, e se questo valore scende a 0 si esegue una signal su una variabile di condizione sulla quale il thread main è in attesa, in questo modo il thread main si accorge che tutti i thread sono terminati e può concludere la fase di chiusura del server, salvando lo stato del server su file e chiudendo il socket.

LA LIBRERIA PER L'UTILIZZO DELL'OS

Viene fornita oltre al server la libreria "libOS.a", che permette al client di sfruttare le funzionalità dell'OS tramite delle semplici chiamate alle funzioni:

- os_connect – per la connessione e la registrazione con l'username specificato
- os_store – per la memorizzazione di un oggetto con il nome specificato (se non presente)
- os_retrieve – per la lettura dell'oggetto corrispondente al nome specificato (se presente)
- os_delete – per l'eliminazione dell'oggetto corrispondente al nome specificato (se presente)
- os_leavgetti di tipo status.

Notare che in caso di fallimento di connessione la funzione os_connect non effettua un secondo tentativo in automatico, è necessario infatti controllare il valore di ritorno (nel client) e chiamare nuovamente la funzione di connessione se necessario.

LIBRERIE DI SUPPORTO

Oltre alla libreria espressamente richiesta sono state utilizzate delle altre librerie di supporto:

- libList.a - contiene l'implementazione di una semplice linked list generica, la quale viene utilizzata come lista di trabocco nella hashtable
- libHashTable.a - contiene l'implementazione di una hashtable generica. Alla creazione della HT viene richiesto oltre alla dimensione della stessa anche una funzione di hashing ed una per confrontare due elementi della HT da utilizzare nelle funzioni di ricerca.
- libHTIterator.a - contiene l'implementazione di un iteratore per l'hashtable, permette quindi di scorrere l'intera HT tramite le funzioni next() ed hasNext().
- libUtils.a - contiene varie funzioni tra cui le funzioni di hash e confronto per semplici stringhe e per oggetti di tipo status.

TESTING

Il target "test" del makefile permette di eseguire un breve esempio di ogni operazione dell'OS.
Lo script test.sh lancia in contemporanea 50 client con nomi fittizi ("user1", user2", …), ognuno dei quali esegue l'operazione di store di 20 file di dimensione crescente fino a 100'000 byte. Ogni file contiene la stringa "0123456789" concatenata più volte.
Una volta terminata l'esecuzione di questo test vengono lanciati altri 50 client utilizzando lo stesso pool di nomi, 30 dei quali eseguono il comando retrieve controllando che i dati ricevuti coincidano con la stringa mostrata prima, e gli altri 20 eseguono la remove degli oggetti salvati in precedenza.

Tutti i client lanciati stampano sul file "testout.log" una riga resoconto del test:

- username;CONNERR – in caso di fallimento durante la connessione al server (i client eseguono 5 tentativi di connessione, con un delay di 3 secondi tra di essi).
- username;STORE;n;t – dove n indica il numero di store andate a buon fine sul numero totale t
- username;RETRIEVE;n;t
- username;DELETE;n;t

A questo punto viene eseguito lo script testsum.sh che riassume il contenuto del file "testout.log" stampando in output il totale dei client, degli errori di connessione e di operazioni che hanno avuto successo rispetto al numero totale.
Infine viene inviato il segnale "SIGUSR1" al server, il quale ne stampa lo stato, ed il segnale "SIGINT" che lo fa terminare.

La compilazione ed il test dell'OS è stato eseguito più volte con esito positivo (senza alcun warning di compilazione, né errore durante l'esecuzione) su:

- Sistema linux Kubuntu 19.04
- Sistema linux Xubuntu 14.10(dal sito del corso) su macchina virtuale in Windows 10