Progetto Sistemi Operativi

Object Store

Autore: Francesco Buti

Data consegna: 31/08/2019

Sommario

Overview:

Il progetto consiste nella realizzazione di un sistema Client-Server, realizzato mediante un'architettura multi-threaded e destinato a fornire funzionalità di deposito, cancellazione e ritiro dati da un sistema di repository interno gestito dal server.

Il server gestisce ogni nuova connessione generando un thread (worker) che si occupa di valutare e soddisfare le richieste del client relativo. Il worker necessita, come argomento, di una struttura in grado di memorizzare il socket, una variabile di mutua esclusione (di cui parleremo in seguito) e una lista dei client attivi. Ogni client può accedere, come richiesto dall'assegnamento, esclusivamente alla propria cartella dati, contrassegnata con il nome del client stesso. Per evitare che più client abbiano accesso concorrente alla stessa cartella, è stata implementata una lista allocata dinamicamente che tenga traccia dei client connessi, alla quale vengono apportate modifiche solo in fase di registrazione (aggiunta del client) e in fase di disconnessione (rimozione del client). La lista contiene in testa un elemento fasullo per semplificarne la gestione. Il nome assegnato è artificiale e si assume che nessun client si connetta con tale nominativo. La testa della lista viene liberata in caso di terminazione del server.

Il server mantiene un insieme di variabili di controllo e statistica aggiornate ad ogni esecuzione, volte a verificare lo stato attuale del sistema (client attualmente connessi, oggetti depositati dall'ultimo controllo, dimensione totale in byte depositati dal server attuale).

In caso di terminazione durante il trasferimento dati, il sistema previene il deposito di file incompleti e termina i client ancora in esecuzione. Tutti i dati temporanei presenti sui socket al momento dell'interruzione vengono scartati. Ogni thread viene realizzato detached, in modo da liberare automaticamente le risorse.

Meccanismi di mutua esclusione:

L'unica struttura dati condivisa da più thread è la lista contenente i nomi delle connessioni attive. Per evitare problemi di concorrenza (che porterebbero anche a potenziali segmentation fault) è stato implementato un meccanismo di mutua esclusione mediante mutex. Al momento della registrazione e della disconnessione, pertanto, ogni client avrà accesso esclusivo alla lista per modificarla.

La variabile mutex usata per questo scopo viene creata nel server, e passata mediante argomento ad ogni thread worker generato per un nuovo client.

Gestione dei segnali:

La gestione dei segnali viene effettuata direttamente all'interno del server. Ogni segnale viene "catturato" tramite *sigwait;* il comportamento varia per ognuno, seguendo le istruzioni dettate nel testo del progetto. In particolare, il sistema gestisce SIGSTP, SIGUSR1 e SIGINT in una semplice funzione. L'unico segnale (lanciabile) in grado di terminare il server è SIGINT, la cui ricezione comporta il settaggio di un flag; questo permette di liberare le risorse allocate dal sistema prima di terminarlo definitivamente mediante l'uso di shutdown.

Protocollo di comunicazione:

Come richiesto dall'assegnamento, client e server comunicano tramite un protocollo basato su stringhe ASCII, il cui scambio avviene mediante lettura e scrittura su socket.

In fase di progettazione è stata scartata la possibilità di leggere a gruppi di un byte; così facendo sarebbe infatti stato possibile ridurre il numero di controlli e operazioni, a discapito però del tempo di esecuzione, che sarebbe stato pesantemente influenzato dall'overhead creatosi. La lettura avviene pertanto a gruppi di 512 byte, una dimensione che permette di ridurre significativamente l'overhead in caso di blocchi di dati di dimensioni comprese tra 100 byte e 100 KB (quali, appunto, quelle perviste per il testing del progetto). In caso di dimensioni più significative, comunque, è facile notare quanto il codice sia velocemente modificabile per adattarsi a quantità di dati più elevate. Lo spreco di memoria per letture inferiori a 512 byte invece, per quanto ovviabile con altri metodi, è comunque trascurabile. La dimensione dei buffer di lettura si basa quindi sulla probabilità di trovarsi in una situazione "media" (non troppe letture di pochissimi byte, non poche letture di file considerevolmente grandi).

Non è stato introdotto un timeout per la lettura dal socket: l'assegnamento non specifica quale sia il comportamento del server nel caso in cui un client ritardi nel completare l'invio di un comando, si assume quindi che rimanga in attesa. Se si presentasse la necessità, la soluzione sarebbe implementabile come segue (ossia tramite un timeout espresso in secondi):

```
struct timeval tv;
tv.tv_sec = timeout_in_seconds;
tv.tv_usec = 0;
setsockopt (sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*) &tv, sizeof tv);
```

Tale soluzione, tuttavia, renderebbe necessario impostare il socket (almeno temporaneamente) come non bloccante.

Durante la lettura e scrittura di blocchi superiori a 512 byte, si fa uso delle funzioni *readn* e *writen,* dichiarate e implementate direttamente nelle librerie lato client e server. Pur essendo sostanzialmente identiche, non è stata creata una libreria condivisa, che sarebbe stata uno spreco di risorse per così poche linee di codice.

Testing:

Il sistema è testabile tramite l'apertura di due terminali nella cartella di lavoro, e il lancio dei comandi *make* (eventualmente preceduto da *make clean* nel caso si rendesse necessario rimuovere i file creati in precedenza), il lancio del server tramite *./server* e, infine, il lancio di *make test* nel secondo terminale. Il risultato dell'esecuzione verrà inserito nel file testout.log, sotto forma di un riassunto sintetico delle operazioni effettuate (riuscite e non riuscite) da parte di ogni singolo client lanciato.

L'uso dello script testsum.sh potrebbe richiedere la modifica delle autorizzazioni mediante chmod.

L'esecuzione di make test comporta il lancio di 100 client (a gruppi di 50) al termine del quale la cartella DATA conterrà 50 cartelle. Tutte le cartelle i cui client hanno eseguito il test 1 e/o 2

presenteranno 20 file di dimensione crescente; nei test vengono depositati file contenenti stringhe da “a” a “t” e in numero costantemente incrementato fino a 100 000, per renderne facilmente verificabile il contenuto.

Tutte le cartelle relative a client che hanno eseguito la terza batteria di test presenteranno solo 19 file, in quanto il lancio del test 3 porta alla cancellazione del secondo file depositato. Sono stati rimossi tutti i messaggi di operazione sul server per garantire una run più “pulita”. L’esecuzione del server con Valgrind evidenzia l’assenza di memory leak alla chiusura.

Il lancio degli script necessita della presenza di un eseguibile funzionante e in esecuzione del server Object-Store.

Di seguito si riporta il risultato dell’esecuzione di un client:

```
---------------------- usertest4 ----------------------

Operations executed :                20
Operations terminated w/ OK :        20
Operations terminated w/o OK [KO] : 0

-------------------------------------------------------
```

Compatibilità:

L’architettura sviluppata è stata testata (e risulta funzionante) sui seguenti sistemi:

- Ubuntu 19.04
- Ubuntu 18.04 LTS
- Ubuntu 16.04 LTS
- Mac OS
- Xubuntu 14.10
- Windows (Subsystem Linux)

Note particolari:

Il progetto è disponibile in forma integrale, come codice open source, al seguente indirizzo: https://github.com/FrancescoButi/SOL. Nel caso in cui si presentassero problemi di perdita o corruzione dei file, si prega di fare riferimento all’indirizzo sopracitato.

Ogni file necessario al funzionamento è stato depositato in un’unica cartella per semplificarne la lettura; la divisione in repository, qualora richiesta, è disponibile nel Git.

La realizzazione del progetto è stata effettuata come singolo, come previsto dall’assegnamento, mediante l’uso di 3 terminali, rispettivamente con sistema Linux a 64 bit, Windows con Subsystem

Linux e Windows con virtual machine. La programmazione del sistema è avvenuta prevalentemente tra giugno e luglio, con modifiche, miglioramenti e correzioni avvenute ad agosto.

Dato lo scambio di informazioni tra client e server, non necessariamente limitato al solo protocollo di comunicazione, l'aggiornamento delle statistiche potrebbe presentare un comportamento diverso in presenza di client o server estranei. Il protocollo, punto centrale della comunicazione, è invece pensato per accogliere anche connessioni da parte di eseguibili non presenti nel sistema consegnato.