`memBox` : un memory storage concorrente

Secondo frammento del progetto 1 del modulo di laboratorio SOL 2016

Contents

1 Introduzione

Il modulo di Laboratorio di Programmazione di Sistema del corso di Sistemi Operativi e Laboratorio (277AA) prevede lo svolgimento di un progetto da sviluppare utilizzando C in ambiente Linux. Il progetto può essere svolto singolarmente dallo studente o al più da due studenti appartenenti allo stesso corso (a meno di differente accordo con i docenti). Questo documento descrive la struttura complessivia del secondo frammento del progetto (scadenza 30/06/2016). Il progetto viene sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentati durante il corso.

1.1 Materiale in linea

Tutto il materiale relativo al corso può essere reperito sul sito web:

`http://didawiki.cli.di.unipi.it/doku.php/informatica/sol/laboratorio16`

Il sito verrà progressivamente aggiornato con le informazioni riguardanti il progetto (es. FAQ, suggerimenti, avvisi, update delle specifiche), il ricevimento e simili. Il sito è un Wiki e gli studenti possono registrarsi per ricevere automaticamente gli aggiornamenti delle pagine che interessano maggiormente. In particolare consigliamo a tutti di registrarsi alla pagina delle 'FAQ' e degli 'Avvisi Urgenti'.

Eventuali chiarimenti possono essere richiesti consultando i docenti del corso durante l'orario di ricevimento, le ore in laboratorio e/o per posta elettronica.

1.2 Struttura del progetto e tempi di consegna

Il progettopresente frammento può essere consegnato dallo studente entro il 30 Giugno 2016.

La consegna del progetto avviene *esclusivamente* inviando per posta elettronica il tar creato dal target `consegna` del `Makefile` contenuto nel kit di sviluppo del progetto. Il tar deve essere allegato da un messaggio con soggetto

`lso16: frammento 2 - corso A/B`

ed inviato all'indirizzo di posta

`lso.di@listgateway.unipi.it`

Non è necessario (e non deve essere fatto) iscriversi alla lista per inviare il messaggio con il progetto. Le consegne sono seguite da un messaggio di conferma (non automatico) da parte di uno dei docenti del corso. La notifica verrà inviata all'indirizzo di posta da cui è pervenuta la consegna del progetto. Se la ricezione non viene confermata entro 3/4 giorni lavorativi, contattare il docente per e-mail.

Si prega di specificare nella mail che allega il progetto, Nome e Cognome dello studente (o degli studenti) che hanno sviluppato il progetto e corso di appartenenza (Corso A o Corso B).

I progetti che non contengono tutti i file necessari o che non compilano ed eseguono correttemante su versioni recenti di Ubuntu x86_64 non saranno accettati. I progetti che non rispettano il formato o il cui tarball non è stato generato con il target `consegna` *non verranno accettati. Si prega di controllare che tutti i file necessari alla corretta compilazione ed esecuzione del progetto siano presenti nel tarball prima di inviarlo.*

La data ultima di consegna è il 30/06/2016. Dopo questa data gli studenti dovranno svolgere il nuovo progetto previsto per il corso 2016 chiamato progetto di recupero che sarà disponibile a inizio luglio 2016.

La consegna del progetto deve comunque essere fatta almeno 15 giorni prima della data dell'appello che si intende sostenere. Fa eccezione solamente il primo appello estivo dell'a.a 15/16, in cui la consegna del progetto può essere fatta una settimana prima della data dell'appello. Si ricorda, che per sostenere l'esame é necessario iscriversi.

1.3 Valutazione dei frammenti del progetto 1

Ai due frammenti viene assegnata una valutazione complessiva da 0 a 30, cui vengono sommati gli eventuali 3+3 bonus se la consegna è stata effettuata entro le scadenze intermedie fornite sul sito web. La *qualità della documentazione allegata* da un contributo fino a 6 punti. La valutazione del progetto è effettuata in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, corretto uso del Makefile e librerie etc.)
- efficienza e robustezza del software
- modalità di testing
- aderenza alle specifiche
- gestione della concorrenza e dei segnali
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione (vedi Sez. 6.3)

La prova orale tenderà a stabilire se lo studente è realmente l'autore del progetto consegnato e verterà su tutto il programma del corso e su tutto quanto usato nel progetto anche se non fa parte del programma del corso. L'esito della prova orale è essenziale per delineare il voto finale. In particolare, l'orale potrà comprendere:

- una discussione delle scelte implementative del progetto
- l'impostazione e la scrittura di semplici script bash e Makefile
- l'impostazione e la scrittura di programmi C + POSIX calls non banali (sia sequenziali che concorrenti)
- domande su tutto il programma presentato durante il corso.

Casi particolari Gli studenti lavoratori iscritti alla laurea triennale in Informatica possono consegnare i due frammenti assegnati in un'unica soluzione ottenendo tutti i bonus in qualsiasi appello dell'AA 2015/16. In questo caso è necessaria la certificazione da consegnare al docente come da regolamento di ateneo.

Gli studenti che svolgono il progetto per le lauree di secondo livello sono invitati a contattare il docente.

Gli studenti iscritti ai vecchi ordinamenti (nei quali il voto di Lab. 4 contribuiva al voto di SO) avranno assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

2 Il progetto: `memBox`

2.1 Descrizione

Lo scopo del progetto è lo sviluppo di un server concorrente che implementa uno spazio di memorizzazione di *oggetti* (*repository*). Un oggetto è una sequenza continua di bytes di dimensione non nulla ed avente una size massima stabilita nel file di configurazione del server `memBox` con l'opzione `MaxObjSize` (vedi Sez. 2.2). L'oggetto è identificato univocamente da una chiave numerica (codificata su un intero a 32 bits). Ogni richiesta inviata al server deve contenere la chiave dell'oggetto a cui la richiesta si riferisce.

Uno o più processi client comunicano con il server per inviare richieste di vario tipo. In particolare le richieste possono essere: di inserimento (PUT_OP), di aggiornamento di un oggetto già inserito (UPDATE_OP), di rimozione di un oggetto presente nel repository (REMOVE_OP), di recupero (GET_OP) del contenuto di un oggetto, di acquisizione della lock (LOCK_OP) sull'intero repository e rilascio della stessa (UNLOCK_OP). È possibile effettuare un'operazione PUT_OP solo per chiavi non presenti nel repository. È possibile effettuare un'operazione UPDATE_OP solo se la size dell'oggetto inviato nella richiesta coincide con la size dell'oggetto memorizzato nel server.

Ad esempio, una richesta di recupero di un oggetto con chiave 100 sarà codificata inviando un messaggio di richiesta al server contenente le informazioni `<100, GET_OP>` opportunamente formattate. Il server risponderà con un messaggio il cui header conterrà l'esito dell'operazione (e.s OP_OK, OP_FAIL, ...), mentre il payload conterrà i bytes dell'oggetto con chiave 100. Se l'oggetto non è presente nel repository (ad esempio perché è stato rimosso da un altro processo), verrà ritornato al client un messaggio di errore opportunamente codificato. Per questo esempio, il messaggio di errore è, ad esempio, `OP_GET_NONE`. Per i possibili codici dei messaggi di risposta inviati dal server al client, vedere il file `ops.h` nel tarball del kit fornito dai docenti.

La comunicazione avviene tramite socket di tipo AF_UNIX. Il path da utilizzare per la creazione del socket AF_UNIX deve essere specificato nel file di configurazione utilizzando l'opzione `UnixPath`. Per rendere più agevole lo sviluppo ed il testing dell'applicazione sulle macchine del centro di calcolo per la didattica, le socket create dallo studente con numero di matricola `matricola` potranno avere un nome che rispetta il seguente formato: `membox_sock_matricola`. L'obiettivo è minimizzare possibili interazioni indesiderate fra progetti sviluppati da utenti diversi sulla stessa macchina del laboratorio.

Le operazioni di lock/unlock si intendono sull'intero repository e non sul singolo oggetto. Se il server ha ricevuto un messaggio di tipo LOCK_OP da un client a cui è associata la connessione n, allora il server dovrà servire solamente le richieste provenienti dalla connessione n (cioè dal client che mantiene la lock). Se il client chiude la connessione, la lock sull'intero repository dovrà essere rilasciata anche se non è stata inviata una esplicita operazione di UNLOCK_OP.

Il server può contenere un numero massimo di oggetti pari a `StorageSize` e non dovrà occupare una size complessiva per gli oggetti maggiore di `StorageByteSize` (se `StorageSize` e/o `StorageByteSize` valgono 0 vuol dire che non c'è nessun limite sul numero massimo di oggetti/dimensione dello storage). Oltre tali valori di soglia, il server non dovrà più accettare richieste di inserimento di nuovi oggetti, mentre dovrà continuare ad accettare altri tipi di richieste (rimozione, aggiornamento, recupero, lock/unlock). Nello sviluppo del codice del server si dovrà tenere in considerazione che:

- il server deve essere in grado di poter memorizzare decine di migliaia di oggetti
- il server deve essere in grado di gestire efficientemente alcune decina di connessioni contemporanee.

Il server al suo interno implementa un sistema multi-threaded che è in grado di gestire contemporaneamente sia nuove connessioni dai client che gestire le richiesti su connessioni già stabilite. Un cliente può sia inviare una richiesta per singola connessione oppure, per una data connessione, inviare più richieste (sia sullo stesso oggetto che su oggetti diversi). Il numero massimo di connessioni concorrenti che il server `memBox` gestisce è stabilito dall'opzione di configurazione `MaxConnections`. Se tale numero viene superato, dovrà essere ritornato al client un opportuno messaggio di errore. Ogni connessione viene gestita interamente da un solo thread. Il numero di thread utilizzati per gestire le connessioni è specificato nel file di configurazione con l'opzioni `ThreadsInPool`. Tale numero (che non può essere 0), specifica la dimensione di un pool di threads che rimangono sempre attivi per tutta la durata dell'esecuzione del server. Il server utilizza i thread del pool per gestire le connessioni dei client. Qualora tutti i thread del pool risultino occupati, possono essere accodate fino a $MaxConnections - ThreadsInPool$ connessioni in attesa che il pool ne serva le richieste. Un thread appartenente al pool, non appena conclude la gestione di una connessione (la socket è stata chiusa), controlla se ci sono connessioni pendenti in coda ed in caso affermativo ne preleva una da gestire. Se non ci sono connessioni in coda, il thread "rientra" nel pool di thread disponibili in attesa di essere svegliato per gestire una nuova connessione.

2.2 File di configurazione

Il server `memBox` viene attivato da shell con il comando

```
$ membox -f membox.conf
```

dove `membox.conf` è il file di configurazione contenente tutti i parametri relativi alla configurazione del server. Il formato del file di configurazione è schematizzato in Fig. 1,

```
# path utilizzato per la creazione del socket AF_UNIX
# ATTENZIONE: se il codice viene sviluppato sulle macchine
#             del laboratorio utilizzare come nome un nome
#             unico, ad esempio:
#             /tmp/mbox_sock_<numero-di-matricola>
UnixPath         = /tmp/mbox_socket

# numero massimo di connessioni pendenti
MaxConnections  = 32

# numero di thread nel pool
ThreadsInPool    = 8

# dimensione dello storage in numero di oggetti (0 nessun limite)
StorageSize      = 0
# dimensione dello storage in numero di bytes (0 nessun limite)
StorageByteSize  = 0

# dimensione massima di un oggetto nello storage (0 nessun limite)
MaxObjSize       = 0

# file nel quale verranno scritte le statistiche
# ATTENZIONE: se il codice viene sviluppato sulle macchine
#             del laboratorio utilizzare come nome un nome
#             unico, ad esempio:
#             /tmp/mbox_stats_<numero-di-matricola>
StatFileName     = /tmp/mbox_stats.txt
```

Figure 1: Formato del file `membox.conf`

Il processo `memBox` termina quando viene ricevuto il segnale `SIGINT` o `SIGTERM` o `SIGQUIT`. Alla ricezione di tale segnale il server esce il prima possibile dopo che ha segnalato a tutti i thread attivi di chiudere immediatamente tutte le connessioni e dopo aver stampato le statistiche nel file associato all'opzione di configurazione `StatFileName`. Tutti i socket unix ed eventuali file temporanei devono essere cancellati all'uscita del server.

2.3 Gestione dei segnali

Oltre ai segnali `SIGINT` o `SIGTERM` o `SIGQUIT` che determinano la terminazione "immediata" di `memBox` , il server gestisce altri 2 segnali: `SIGUSR1` e `SIGUSR2`. Se viene

inviato un segnale di tipo `SIGUSR1` il server stampa le statistiche correnti nel file associato all'opzione `StatFileName` presente nel file di configurazione secondo il formato descritto nella sezione seguente. Se tale opzione è vuota o non è presente, il segnale ricevuto viene ignorato. Se viene inviato un segnale di tipo `SIGUSR2` il server smette di accettare nuove connessioni in ingresso e, prima di terminare, aspettare che tutte le connessioni in corso vengano chiuse senza forzare l'uscita immediata di tutti i threads. Prima di uscire il server stampa le statistiche nel file associato all'opzione `StatFileName`.

2.4 Statistiche

Alla ricezione del segnale `SIGUSR1` (e prima di terminare) il server appende nel file `StatFileName` le statistiche del server. Le stampa nel file deve avere il formato `<timestamp - statdata>` dove `timestamp` è l'istante temporale in cui le statistiche vengono inserite nel file e `statdata` è una sequenza di numeri (separati da uno spazio) che riportano i valori monitorati durante il tempo di vita del server. I dati statistici minimi che devono essere prodotti nel file sono riportati in Fig. 2. Se lo studente lo ritiene, può arricchire le statistiche prodotte documentandole opportunamente nella relazione che accompagna il progetto.

```
n. di PUT_OP ricevute       n. di PUT_OP fallite
n. di UPDATE_OP ricevute    n. di UPDATE_OP fallite
n. di GET_OP ricevute       n. di GET_OP fallite
n. di REMOVE_OP ricevute    n. di REMOVE_OP fallite
n. di LOCK_OP ricevute      n. di LOCK_OP fallite
n. di connessioni concorrenti
n. massimo di oggetti memorizzati raggiunto
n. corrente di oggetti memorizzati
size massima in bytes raggiunta
size in bytes corrente
```

Figure 2: Dati statistici minimi che devono essere prodotti.

2.5 Protocollo client server

Il protocollo di comunicazione per ogni tipo di richiesta che il client può inviare al server deve essere stabilito dallo studente e documentato nella relazione. Per effettuare i test deve essere utilizzato il client (`client.c`) fornito dai docenti. In particolare, devono essere definiti chiaramente i tipi dei messaggi di richiesta e risposta e come sono gestiti eventuali messaggi di errore che il server ritorna al client (ad esempio a seguito di una operazione di tipo GET_OP di un oggetto non esistente nel repository).

3 Script bash

Deve essere realizzato un script bash che dato in input il file delle statistiche produca in output opportune informazione sulla base dei parametri passati allo script.

Lo script si attiva da linea di comendo con diverse opzioni

```
mboxstat.sh statfile
mboxstat.sh statfile -p -u -g -r -c -s -o -m
mboxstat.sh --help
```

Senza nessuna opzione lo script stampa tutte le statistiche dell'ultimo timestamp. L'opzione -p stampa il numero di PUT_OP ed il numero di PUT_OP fallite per tutti i timestamp. L'opzione -u stampa il numero di UPDATE_OP ed il numero di UPDATE_OP fallite per tutti i timestamp. L'opzione -g stampa il numero di GET_OP ed il numero di GET_OP fallite per tutti i timestamp. L'opzione -r stampa il numero di REMOVE_OP ed il numero di REMOVE_OP fallite per tutti i timestamp. L'opzione -c stampa il numero di connessioni per tutti i timestamp. L'opzione -s stampa la size in KB per tutti i timestamp. L'opzione -o stampa il numero di oggetti nel repository per tutti i timestamp. L'opzione -m stampa il massimo numero di connessioni raggiunte dal server, il massimo numero di oggetti memorizzati ed la massima size in KB raggiunta. Infine l'opzione lunga `help` stampa un messaggio di uso dello script.

Le opzioni possono apparire in un qualsiasi ordine sulla linea di comando e più opzioni possono essere usate contemporaneamente. È fatto esplicito divieto di usare `sed` o `awk` nella realizzazione dello script.

Come esempio di output dello script, si consideri il seguente caso:

```
> cat /tmp/mbox_stats.txt
1462250353 - 7027 3 152 8 875 8 7112 102 11 0 3 1136005 13729157 17 1043
1462250358 - 7031 3 155 8 879 8 7120 105 12 0 0 148351 13729157 16 1043
1462250358 - 7044 3 164 8 885 8 7130 109 12 0 7 3280021 13729157 23 1043
1462250358 - 7044 3 164 8 885 8 7131 110 12 0 7 3280021 13729157 23 1043
1462250358 - 7044 3 164 8 885 8 7132 111 12 0 7 3280021 13729157 23 1043
1462250358 - 7044 3 164 8 885 8 7133 112 12 0 7 3280021 13729157 23 1043

> ./memboxstat.sh -p -r /tmp/mbox_stats.txt
# timestamp            PUT    PUTFAILED        REMOVE REMOVEFAILED
1462250353            7027            3          7112          102
1462250358            7031            3          7120          105
1462250358            7044            3          7130          109
1462250358            7044            3          7131          110
1462250358            7044            3          7132          111
1462250358            7044            3          7133          112
```

4 Istruzioni

4.1 Materiale fornito dai docenti

Nei kit del progetto vengono forniti

- `client` da usare per testare le interazioni con il server
- `Makefile` per test e consegna. Il Makefile dovrà essere modificato nelle parti indicate
- alcuni file di intestazione (.h) con definizione dei prototipi e delle strutture dati
- una implementazione di una hash-table (file icl_hash.c ed icl_hash.h) che può essere usata dallo studente
- vari `README` di istruzioni

4.2 Cosa devono fare gli studenti

Gli studenti devono:

- leggere *attentamente* le specifiche del progetto (questo documento), i README e capire il funzionamento del codice fornito dai docenti (in particolare il client)
- implementare tutte le funzionalità richieste per il server

- verificare le funzioni con i test forniti dai docenti (attenzione: questi test vanno eseguiti su codice già corretto e funzionante altrimenti possono dare risultati fourvianti o di difficile interpretazione)
- preparare la *documentazione*: vedi la Sez. 6.
- sottomettere il progetto finale esclusivamente utilizzando il Makefile fornito e seguendo le istruzioni riportate nel README.

5 Parti Opzionali

Possono essere realizzate funzionalità ed opzioni in più rispetto a quelle richieste (ad esempio aggiungendo altre opzioni nel file di configurazione aumentando le funzionalità del server, inserimento di nuove statistiche, fornire grafici per le statistiche etc.).

Le parti opzionali devono essere spiegate nella relazione e corredate di test appropriati ove opportuno.

6 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

6.1 Vincoli sul codice

Makefile e codice devono compilare ed eseguire CORRETTAMENTE su versoni recenti di Ubuntu x86_64 o su un insieme non vuoto delle macchine del Centro di Calcolo (CDC). La relazione deve specificare su quali versioni di Ubuntu o su quali macchine CDC è possibile far girare correttamente il codice.

La stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire attraverso il Makefile fornito dai docenti
- il codice deve compilare senza errori o warning gravi utilizzando le opzioni `-Wall -pedantic`
- NON devono essere utilizzate funzioni di temporizzazione quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi/thread. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi/thread coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)
- il server non deve generare memory leaks (testare l'esecuzione del server con valgrind)

6.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.

- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne (se non ovvio dal contesto)

6.3 Relazione

Per la consegna finale del progetto didattico (composto dal frammento 1 e dal frammento 2) lo studente deve preparare una breve relazione (massimo 15 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La documentazione del progetto consiste nei commenti al codice e nella relazione. La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi. In particolare, NON devono essere ripetute le specifiche contenute in questo documento.* In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e loro motivazioni)
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- la struttura dei programmi sviluppati
- la struttura dei programmi di test aggiuntivi oltre a quelli forniti dai docenti
- l'interazione fra i processi/threads ed i relativi protocolli
- le difficoltà incontrate e le soluzioni adottate
- eventuali estensioni che lo studente ha sviluppato oltre a quanto richiesto
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- README di istruzioni su come compilare/eseguire ed utilizzare il codice

La relazione deve essere in formato PDF.