Angelo Catalani 1582230

RELAZIONE DEL PROGETTO DI SISTEMI OPERATIVI (6 CFU).

Introduzione.

Specifica :

Realizzazione di un servizio di scambio messaggi supportato tramite un server sequenziale o concorrente (a scelta). Il servizio deve accettare messaggi provenienti da client (ospitati in generale su macchine distinte da quella dove risiede il server) ed archiviarli.

L'applicazione client deve fornire ad un utente le seguenti funzioni:
1. Lettura tutti i messaggi spediti all'utente.
2. Spedizione di un nuovo messaggio a uno qualunque degli utenti del sistema.
3. Cancellare dei messaggi ricevuti dall'utente.

Un messaggio deve contenere almeno i campi Destinatario, Oggetto e Testo.
Si precisa che la specifica prevede la realizzazione sia dell'applicazione client ce di quella server. Inoltre, il servizio potrà essere utilizzato solo da utenti autorizzati (deve essere quindi previsto un meccanismo di autenticazione).

Plus :

Il client fornisce anche le seguenti funzioni:

1. Lettura tutti i messaggi spediti dall'utente.
2. Cancellare dei messaggi inviati all'utente.
3. Visualizzare utenti on-line.
4. Procedura di registrazione.
5. Visualizzare messaggi rimossi.

Il server non consente l'autenticazione ad uno stesso utente già autenticato.

Implementazione.

Scelte di progetto e realizzative:

Il tipo di comunicazione implementato attraverso socket è :

1. protocollo:TCP
2. comunicazione:sock_stream

Il server è multi-thread: il main thread è in attesa di una nuova connessione ed in caso di successo lancia un nuovo thread per la gestione della connessione per poi rimettersi in attesa.

Il server può essere lanciato con i parametri : setup oppure nosetup.
Nel primo caso si auto configura :

1. Creazione della cartella “msgbox” per contenere nuovi messaggi .
2. Creazione dei file “msg_box” e “users” per tenere traccia dei messaggi scambiati e degli utenti registrati.
3. Nel caso in cui siano già presenti alcuni di questi file (o directory) provvede ad eliminarne il contenuto.

L’implementazione di quest’ultimo step avviene attraverso una funzione ricorsiva che elimina il contenuto anche di eventuali sub-directory annidate.
Per distinguere un regular-file da un directory-file ho utilizzato le libreria stat.h :la funzione stat e le macro S_ISREG e S_ISDIR
Se il parametro è nosetup il server usa l’ultima configurazione e ci saranno errori inevitabili nel caso in cui il server non sia stato precedentemente configurato.

Il server per la gestione degli utenti usa la struttura dati “msg_box” caratterizzata da:

1. Utenti registrati:hash-table le cui collisioni sono implementate attraverso una lista collegata.
2. Numero totale di utenti
3. Numero totale di messaggi

La msg_box è inizializzata dal main thread e condivisa da tutti i thread che gestiscono eventuali connessioni.
La scelta di un server multi-thread rispetto ad uno multi-processo si giustifica nella semplicità di realizzare la condivisione della struttura dati sopra citata.
Ogni accesso alla msg_box è realizzato in mutua esclusione attraverso un unico semaforo binario.
Non è stato possibile realizzare una sincronizzazione a grana più fine poiché nel caso in cui l’hash table in seguito ad un operazione di add si fosse allargata(nel caso in cui il load-factor divenisse>=0.7) ci sarebbero stati possibili deadlock se avessi adottato un semaforo binario per ogni cella del bucket array dell hash table.

Il server setta a 100 secondi la massima attesa in seguito ad una wait per riconoscere situazioni di deadlock , le quali sono tuttavia teoricamente impossibili.

Il server in caso di shut-down (ricezione dell evento SIGINT) , prima di liberare la memoria allocata provvede ad assicurarsi che nessun thread sia in sezione critica,al fine di mantenere l’integrità dei dati presenti su storage fisico permanente.

Un thread lanciato dal server , nel caso in cui fosse in attesa per piu di 1000 secondi a causa di un operazione bloccante termina:tale situazione impedisce che un client sia inattivo per più di 1000 secondi.

Il main thread prima di accettare connessioni alloca la msg_box , la lista degli utenti online,apre i due file (“msg_box” e “users”) i cui descrittori sono salvati in variabili globali per consentire l’accesso sempre in mutua esclusione ai thread, ed inizializza i semafori. Poichè le operazioni sulla struttura dati condivisa coinvolgono anche accesso a file , la mutua esclusione è comunque garantita dall’utilizzo dello stesso semaforo sopra citato.

La lista degli utenti online implementata come lista collegata di nomi utente(stringhe) è condivisa in mutua esclusione tra i thread attraverso un secondo semaforo binario. Quest’ultima è consultata per visualizzare utenti online (lato client)ed impedire l’autenticazione a più applicazioni client con uno stesso nome utente(lato server).
Ogni utente è caratterizzato da:

1. nome:stringa
2. password:stringa
3. messaggi ricevuti/inviati/eliminati : 3 liste collegate di stringhe

I nomi degli utenti sono univoci: non è consentita la registrazione con uno stesso nome
Il file “users” è cosi strutturato, ogni riga contiene:

1. nome utente.
2. password.

La registrazione di un utente comporta l’aggiunta di una nuova riga nel file “users”.

Il file “msg_box” è cosi strutturato,ogni riga contiene:

1. nome utente che invia un messaggio
2. nome utente che riceve il messaggio
3. nome del file contenente il testo del messaggio
4. flag che può essere :
   - 0 se il messaggio è presente nella lista dei messaggi inviati dall’utente emittente e nella lista dei messaggi eliminati dal destinatario
   - 1 se il messaggio è presente solo nella lista dei messaggi ricevuti dall’utente destinatario e nella lista dei messaggi eliminati dall’emittente
   - 2 se il messaggio è presente nella lista dei messaggi inviati dall’utente emittente e nella lista dei messaggi ricevuti dal destinatario
   - 3 se il messaggio è presente nella lista dei messaggi eliminati dall’utente emittente e nella lista dei messaggi eliminati dal destinatario

L’uso di tale flag risponde all’esigenza di tenere traccia dei messaggi rimossi e quindi di individuare i messaggi rimossi sia dall’emittente che dal destinatario (flag = 3) che devono quindi essere eliminati (operazione di unlink) dallo storage fisico.
Infatti i messaggi presenti nella lista trash non possono essere letti ma è possibile che siano presenti ancora in memoria secondaria (flag=0 o flag=1).
La creazione di un messaggio comporta l’aggiunta di una nuova linea nel file “msg_box” ove il nome del messaggio è un valore progressivo che si ottiene consultando il numero di messaggi totali scambiati.
La creazione fisica di un file contente il contenuto richiesto dall’utente avviene nella directory chiamata ”msg_log”.

La gestione degli errori comuni , dell’evento di broken-pipe , mancata ricezione di un messaggio perché o il client o il server si disconnettono e l’errore EAGAIN (in caso del

timeout di 1000s) avviene attraverso 3 diverse funzioni (3 per il server e 3 per il client)che eventualmente causano la terminazione solo del thread e non del main thread se invocate dal server.
In particolare per la gestione dell'evento di broken-pipe ho definito un funzione specifica di invio che sfrutta la sys call send con il flag MSG_NOSIGNAL per gestire successivamente l'eventuale errore diEPIPE.

Il server per fornire i servizi si avvale di un modulo esterno con funzioni di manipolazione della msg_box.
All'interno di esse vi sono quelle che producono un risultato in output che però deve essere trasmesso al client.
Nell'ipotesi in cui il server e client risiedano nella stessa macchina una possibile soluzione sarebbe stata :

1. client comunica al server il nome del suo terminale ,ossia la stringa ottenuta dalla chiamata ttyname(1=stdoutput file descriptor)
2. il server esegue una open su tale stringa e successivamente redireziona il suo standard output sul descrittore ottenuto dalla open precedente.
3. il server resetta lo stato iniziale dello standard output .

Tuttavia nel caso corrente ho adottato la seguente soluzione:

1. server re-direziona lo standard output sul descrittore del socket di comunicazione.
2. server resetta lo stato precedente dello std output .

Quest'ultimo step richiede di ottenere ,prima di eseguire lo step1 , una copia dello std output attraverso la chiamata int f1=dup(1) ed una volta eseguito lo step2 per ristabilire la condizione iniziale eseguire dup2(f1,1) e quindi close(f1).

Le operazioni di invio e ricezione sia lato client che server sono implementate attraverso l'utilizzo del carattere di '\n' come terminatore. In entrambe le funzioni vi è la gestione dei residui e dell'evento EINTR.
Nel caso della creazione di un messaggio il cui testo può richiedere più righe il client prima di inviare al server l'input dell'utente contenente il testo del messaggio gli comunica il numero di righe del messaggio ,che equivalgono al numero di chiamate a myRecv che il server dovrà fare per acquisire tutto il testo.
L'operazione di ricezione salva l'output facendo side-effect sul parametro di tipo char* che gli viene passato. Al fine di un maggiore controllo di possibili memory-leak tale parametro è allocato sulla stack e non nell'heap.

Il server non genera memory-leak se prima di invocare l'evento di SIGINT vengono fatti terminare i suoi thread. Altrimenti gli unici memory-leak sono legati agli argomenti passati come parametri ai thread che devono essere necessariamente allocati in stack per evitare race-condition.In alternativa avrei potuto utilizzare un array globale sovradimensionato ove salvare in ogni cella gli argomenti dei thread. Tuttavia ciò avrebbe limitato il grado di multiprogrammazione.
Il client invece non genera memory-leak in ogni caso .
La gestione della struttura dati msg_box , dell'hash table e della lista concatenata è realizzata attraverso 3 diversi moduli da me implementati.
L'utilizzo corretto di tali moduli non causa memory-leak

Ho utilizzato valgrind come strumento di memory-check.

Il client infine acquisisce l'input dall'utente attraverso la funzione fgets , usando stdin come argomento di tipo FILE*.
Tale scelta evita possibili buffer-overflow con la limitazione a 1024 come massima dimensione del buffer di input.

Manuale d'uso.

Compilare:

Ho scritto un makefile che sovraintende la compilazione di tutti i moduli necessari per la generazione degli eseguibili :server e newClient

Installare:

Il server deve essere lanciato o con il parametro setup o con nosetup.
Il parametro deve essere setup se:

1. è la prima volta che si esegue su una macchina il server
2. si vuole resettare il database gestito dal server

Il parametro deve essere nosetup se:

1. si vuole lo stato del database relativo all'ultima esecuzione del server su quella macchina.

Sorgenti.

Sorgenti:

I sorgenti sono :

1. list.c
2. hash-table.c
3. utilities.c :gestione della msg_box
4. client.c
5. client_utilities.c
6. server.c
7. server_utilities.c