Funzionamento generale

Client:

Al client da linea di comando possono essere passate varie opzioni. Se è presente l'opzione -h il client stampa sullo standard output le istruzioni d'uso e termina (indipendentemente dalle altre opzioni eventualmente presenti). Se l'opzione -h non è presente allora è necessaria una ed una sola opzione -f seguita dal nome del socket su cui tentare la connessione, altrimenti il programma termina. Opzionalmente possono essere presenti altre opzioni (se non sono presenti il client tenta la connessione sul socket specificato e termina appena la connessione ha successo o quando scade il tempo di attesa per la connessione). Ogni opzione corrisponde a una richiesta al server, che potrà però essere costituita da più messaggi scambiati secondo il protocollo richiesta-risposta, ad esempio l'opzione -r file1,file2,file3 costituisce una singola richiesta ma col server vengono scambiati più messaggi
(ciò assume rilevanza se è specificata l'opzione -t che specifica il tempo che intercorre tra due richieste). Poiché il server memorizza i file con il loro path assoluto, il client, qualora gli vengano forniti path relativi, li traduce in path assoluti (basandosi sulla cwd del client) prima di inviare la richiesta al server. Per comunicare col server il client invia prima un messaggio di lunghezza nota dove specifica il tipo di operazione e la lunghezza del/dei messaggi seguenti, e poi invia al server gli argomenti necessari per svolgere l'operazione (ad esempio il path del file) in accordo con quanto specificato nel primo messaggio. Una volta effettuato l'invio il client si mette in ascolto di una risposta da parte del server che sarà costituita da un messaggio di lunghezza nota (nello stesso formato del primo messaggio inviato) dove sarà riportato l'esito dell'operazione e la lunghezza di eventuali messaggi successivi con i quali il server invierà se necessario ciò che era stato richiesto dal client (ad esempio il file in una richiesta di read). L'unica eccezione è la richiesta -R durante la quale (poiché il numero di file che verranno restituiti non è noto a priori) il server informerà il client, prima di inviare ogni file, della dimensione di quest'ultimo. Qualora fosse specificata
l'opzione -d il cliente salverà il/i file ricevuti da server nella cartella specificata utilizzando solo il nome del file (e scartando quindi tutto il path), se quindi vengono restituiti dal server due file con lo stesso nome ma path diverso il primo dei due ad essere salvato verrà sovrascritto dall'altro (vengono sovrascritti anche eventuali file già presenti nella cartella con lo stesso nome di uno dei file restituiti dal server). La comunicazione del client col server è implementata utilizzando l'API fornita che traduce gli argomenti passati alle funzioni in messaggi che vengono poi scritti sul socket (se la connessione è aperta, altrimenti si ritorna un errore che verrà stampato sullo standard error se l'opzione -p è specificata).

Server:

Il server è composto da un thread master, n thread worker e un thread per la gestione dei segnali. Il file di configurazione (da passare a linea di comando all'avvio) deve specificare il numero di lavoratori (N_WORKER), la dimensione massima dello storage in bytes (MAX_STORAGE), il numero massimo di file (MAX_N_FILE), la dimensione della tabella di hash (si veda in seguito) per la memorizzazione dei file (N_BUCKETS) e il nome del socket (SOCKET_NAME), nel formato "<NAME> = <value>", uno per riga e nell'ordine specificato. Se si verificano errori nella lettura del file di configurazione o se il file non viene fornito vengono usati dei valori di default. La memorizzazione dei file nello storage avviene in memoria principale utilizzando una tabella hash che garantisce rapidità negli inserimenti, ricerche e cancellazioni. All'avvio il server maschera opportunamente i segnali, crea il thread per la gestione dei segnali, legge il file di configurazione e sulla base dei valori letti (o quelli di default) crea il socket su cui accettare le connessioni, inizializza lo storage e crea il pool di thread. Il thread main a questo punto avrà il solo compito di ascoltare e accettare le connessioni in ingresso che verranno opportunamente passate ai thread worker, finché il thread che gestisce i segnali non notifica l'arrivo di uno dei 3 segnali per la terminazione del server. La comunicazione dal thread main ai thread worker avviene attraverso un'unica coda condivisa, mentre la comunicazione dai thread worker al thread main avviene attraverso un'unica pipe. Un'ulteriore pipe (distinta dalla precedente) è utilizzata dal thread che gestisce i segnali per "risvegliare" il thread main dall'attesa di nuove connessioni e informarlo che il segnale è arrivato. Un thread worker una volta avviato esegue in loop la sua routine: legge dalla coda condivisa il "lavoro" da eseguire (se la coda è vuota il thread si mette in attesa che un inserimento venga effettuato), in base a ciò che viene letto dalla coda esegue l'operazione corrispondente e ricomincia da capo. Il thread esce da questa routine quando legge dalla coda un particolare "lavoro" che il thread main inserisce per avvisare i thread worker che un segnale di terminazione è arrivato, nello specifico il thread main inserisce un solo lavoro di questo tipo che poi una volta letto da un thread viene reinserito nella coda e quindi viene propagato a tutti i thread. Fatta esclusione per questo caso particolare i lavori che un worker può leggere dalla coda possono essere di due tipi: la lettura di un nuovo messaggio di un client o un lavoro che non era stato possibile portare a termine in precedenza, per via di una

lock su un file, ma che potrebbe essere compiuto in quanto la lock su tale file è stata rilasciata (il funzionamento delle operazioni di lock e unlock è descritto in dettaglio in seguito).

Operazioni implementate

Di seguito sono riportate le operazioni che il server (nello specifico i worker) può eseguire.
Apertura di un file: se il flag O_CREATE è settato il server cerca il file col nome specificato, se esiste già viene inviato un messaggio al client con l'errore corrispondente, altrimenti il file viene creato (e inizializzato) e inserito nella tabella hash. Come scelta implementativa un file creato (che quindi ha dimensione 0) non viene contato nel numero di file presenti nel server ne viene rimosso dall'algoritmo di rimpiazzamento o restituito nella lettura di più file finché la sua dimensione non aumenta (a seguito di una write o un'append). Se invece il flag O_CREATE non è settato e il file non esiste viene inviato un messaggio al client con l'errore corrispondente, altrimenti, se nessuno possiede la lock sul file, viene aperto e quindi il client sarà autorizzato ad eseguire le varie operazioni su di esso (non ci sono errori se un client apre più volte lo stesso file, ma il file andrà chiuso una volta sola). Se il flag O_LOCK è settato la lock viene acquisita sul file in maniera atomica (non possono quindi intercorrere altre operazioni tra l'apertura del file e l'acquisizione della lock). Se l'apertura del file ha successo il client viene informato con un messaggio di conferma.
Lettura di un file: il contenuto del file corrispondente al path specificato viene inviato al client. Per eseguire questa operazione è necessario aver aperto il file in precedenza (altrimenti al client viene inviato un messaggio di errore) e nessun altro deve possedere la lock su tale file.
Lettura di più file: vengono inviati al client il numero specificato di file (o tutti i file se il numero non è specificato) presenti nel server e sui quali nessuno (a parte il client che ha eseguito la richiesta) possiede la lock. Possono essere inviati meno file rispetto al numero specificato se non ci sono abbastanza file. L'ordine in cui i file vengono inviati dipende da come sono memorizzati nella tabella hash, non è possibile quindi da parte del client alcuna assunzione su quali file riceverà.
Scrittura di un file: scrive il contenuto di un messaggio del client in un file precedentemente creato. È necessario che sul file creato non siano state effettuate altre operazioni (con successo) prima della write, ed è inoltre necessario possedere la lock su tale file, quindi è possibile eseguire questa operazione solo se precedentemente il file è stato creato con il flag O_LOCK settato o se la lock è stata acquisita prima che il file venisse modificato da altri client. Se viene eseguita senza rispettare tali precondizioni viene inviato al client un messaggio di errore.
Scrittura in append: appende ad un file il contenuto di un messaggio del client. Per eseguire questa operazione è necessario aver aperto il file in precedenza (altrimenti al client viene inviato un messaggio di errore) e nessun altro deve possedere la lock su tale file. Come nel caso della write viene controllato se il file può ancora essere memorizzato nel server e in caso negativo viene inviato un messaggio con l'errore corrispondente al client. Se il file può essere memorizzato ma non c'è abbastanza spazio libero viene eseguito l'algoritmo di rimpiazzamento che elimina tanti file quanti sono necessari per fare spazio al nuovo file.
Acquisizione lock: acquisisce la lock su di un file aperto in precedenza.
Rilascio lock: rilascia la lock di un file aperto in precedenza e sul quale era stata acquisita la lock o che era stato creato con il flag O_LOCK. Se si tenta di rilasciare la lock su un file di cui non si possiede la lock viene inviato un messaggio di errore.
Chiusura di un file: chiude un file precedentemente aperto, non sarà quindi più possibile compiere operazioni su tale file a meno che non venga nuovamente aperto. Se il client che richiede la chiusura detiene la lock sul file, la lock verrà rilasciata automaticamente. Un tentativo di chiusura di un file che non è aperto restituisce un errore al client.
Rimozione di un file: segna un file come rimosso, ma non lo rimuove effettivamente finché tutti i client che avevano quel file aperto non lo chiudono. Non sarà però possibile aprire tale file per i client per i quali il file non era aperto al momento della rimozione, se un client tenta di aprire un file che è stato rimosso (anche se è ancora aperto da almeno un client) viene informato con un messaggio di errore che il file non esiste. Se un client tenta di creare un file con lo stesso nome di un file rimosso che però è ancora aperto da qualcuno riceve un messaggio di errore.

Corrispondenza tra opzioni del client e operazioni del server

Non vi è una corrispondenza uno a uno tra le opzioni che il client riceve da linea di comando e le richieste che il client fa al server (tramite l'API). In particolare:

-W: corrisponde ad una'operazione di apertura con i flag O_LOCK e O_CREATE settati, una di scrittura e una di chiusura per ogni file specificato.
-w: si eseguono tre operazioni (apertura, scrittura, chiusura) come nel caso di -W per ogni file nella cartella specificata (e nelle sottocartelle) finché non si raggiunge il numero di file se specificato. Il numero specificato dopo il nome della cartella indica il numero di file da provare a scrivere nel server, se un'operazione dovesse fallire (ad esempio perché il file è già presente nel server) verrà comunque contata per il raggiungimento del numero specificato.
-r: si esegue un'operazione di apertura senza flag settati, una di lettura e una di chiusura per ogni file specificato. Il risultato dell'operazione di lettura viene salvato nella cartella specificata con l'opzione -d se presente.
-R: corrisponde ad un'unica operazione di lettura di più file. I risultati della richiesta vengono salvati nella cartella specificata con l'opzione -d se presente.
-l: corrisponde a un'operazione di apertura seguita da una di lock (le due operazioni non sono atomiche quindi non equivale a una richiesta di apertura con il flag O_LOCK settato).
-u: corrisponde ad un'unica operazione di unlock.
-c: corrisponde ad un'unica operazione di rimozione.
Non è necessario rispettare alcun ordinamento nelle opzioni passate a linea di comando al client tranne che per l'opzione -d che deve trovarsi dopo (non necessariamente subito dopo) un opzione -r o -R.

Politica di rimpiazzamento

Se una delle operazioni di scrittura causa il superamento del limite di memoria o numero di file del server viene utilizzato l'algoritmo di rimpiazzamento (per selezionare un file vittima da eliminare dal server) finché entrambi i limiti non saranno nuovamente rispettati. L'algoritmo è implementato utilizzando una politica LRU cercando però di preservare i file sui quali è detenuta la lock. L'algoritmo quindi elimina il file utilizzato (cioè su cui è stata fatta una qualsiasi operazione con successo) meno recentemente tra i file sui quali nessuno detiene la lock, se non esistono file senza lock allora viene eliminato quello utilizzato meno recentemente. L'eliminazione di un file da parte dell'algoritmo è istantanea (a differenza dell'operazione di rimozione), perciò eventuali operazioni che verranno tentate su di un file eliminato falliranno con un messaggio di errore inviato al client che lo informa che il file non esiste (anche se il client aveva già aperto il file in precedenza). I file eliminati dall'algoritmo sono eliminati permanentemente e non vengono inviati al client che ne ha causato l'eliminazione in quanto l'opzione -D non è implementata.

Meccanismo di lock/unlock

Se una delle operazioni non dovesse essere eseguibile perché la lock è detenuta da un altro client, la richiesta verrà messa in una coda di attesa relativa al singolo file (ogni file ha quindi la sua coda di attesa) e rimarrà lì finché la lock non verrà rilasciata. Il thread che esegue l'operazione che rilascia la lock sposterà tutte le eventuali richieste in attesa presenti nella coda condivisa (la stessa dove il thread main inserisce la notifica che è possibile leggere dati in ingresso da una connessione) affinché l'esecuzione di tali operazioni possa essere tentata nuovamente, anche se non c'è alcuna garanzia che vengano effettuate per prime (in una situazione in cui la coda condivisa non è vuota, poiché gli inserimenti vengono effettuati in coda una delle richieste già presenti nella coda potrebbe richiedere di acquisire la lock sul file). Essendo implementato in questo modo, il meccanismo delle lock non blocca il thread worker che esegue l'operazione nel caso ci sia da attendere. Non viene però effettuato alcun controllo sulle richieste di lock dei client che potrebbero quindi portare a situazioni di deadlock nelle quali i server continua a essere operativo ma i file interessati dal deadlock non sono più accessibili e non possono neanche essere rimossi se non dall'algoritmo di rimpiazzamento nel caso in cui il limite di memoria o del numero di file venga superato e anche su tutti gli altri file sia detenuta la lock (sarebbero possibili varie soluzioni come ad esempio un tempo massimo dopo il quale la lock viene automaticamente rilasciata, una lunghezza massima della coda di attesa di un file superata la quale la lock viene rilasciata o l'imposizione ai client di specificare i file di cui hanno bisogno all'inizio della connessione per poter utilizzare l'algoritmo del banchiere nell'assegnamento delle varie lock, che però non sono implementate nel progetto).
Quando un client si disconnette dal server (anche nel mezzo di uno scambio di messaggi col server) tutti i file che erano aperti vengono chiusi (ed eventualmente cancellati se era stata fatta richiesta della loro rimozione e il client che si è disconnesso era l'unico ad averli aperti).

Gestione della concorrenza fra i thread worker

Per gestire la concorrenza fra i vari worker all'interno del server si utilizza il protocollo lettori/scrittori (fair per entrambi) che garantisce che le operazioni di lettura e scrittura vengano effettuate in mutua esclusione (con uno scrittore alla volta o più lettori contemporaneamente) e che nessun thread debba aspettare indefinitamente prima di poter svolgere la sua operazione. Per operazioni di lettura si intendono tutte le operazioni che non modificano lo storage del server (fatta eccezione per il tempo di ultimo accesso di un file per il quale è presente un'opportuna mutex) quindi solo l'operazione di lettura di un file e di lettura di più file, le operazioni di scrittura invece sono tutte le altre.

Dettagli implementativi

Struttura delle cartelle

Il progetto è composto da più file sorgente, che si trovano nella cartella "src" sia per il client che per il server. Nella cartella include troviamo i file di include e nelle cartelle "obj" e "lib" verranno salvati, dopo aver compilato con il Makefile, rispettivamente i file oggetto relativi ai file sorgente e le librerie. La cartella "config" contiene i file di configurazione per il server. Nella cartella "random_file" si trovano vari file di varie dimensioni utilizzati durante il testing mentre la cartella "dest" che è inizialmente vuota serve per ospitare i risultati delle letture del client durante i test. Il Makefile oltre a generare gli eseguibili, che verranno salvati nella cartella "bin", i file oggetto e le librerie possiede i phony target "clean" per rimuover solo i file eseguibili, "cleandest" per rimuovere tutti i file nella cartella "dest" (utile se si vuole ripetere più volte un test), "cleanall" per rimuovere tutti i file generati dal Makefile o dall'esecuzione di test, "test1" e "test2" per l'esecuzione dei due test che utilizzano rispettivamente gli script "test1.sh" e "test2.sh".

Suddivisione in file

Di seguito sono spiegati le funzionalità coperte dai vari file:
utils.h: un header che include la maggior parte dei file di include necessari sia per il server che per il client, vengono inoltre definite delle macro utilizzate frequentemente nel client.
i_conn.h: header relativo alla libreria i_conn.so (che implementa l'API) per la connessione al server, oltre a dichiarare le funzioni dell'API definisce delle macro per la ricezione e l'invio di messaggi lato client, il valore dei due flag O_CREATE e O_LOCK e definisce lo struct che costituisce il primo messaggio di lunghezza nota (una definizione identica è presente in una libreria del server).
conn_supp.h: header per la libreria conn_supp.so (necessaria per il corretto funzionamento della libreria i_conn.so), in questo file vengono solo dichiarate le funzioni presenti nella libreria.
server_lib.h: include alcune delle librerie precedenti e altre librerie per il funzionamento del server, definisce i due flag O_CREATE e O_LOCK (con gli stessi valori del client), definisce i valori di default da usare ne caso non si fornisca il file di configurazione, definisce delle macro per la ricezione e l'invio di messaggi lato server e definisce vari struct per il salvataggio dei file in memoria principale, per il passaggio di argomenti ai thread e per l'ivio di messaggi (viene definita qui la struct per il primo messaggio come nel client).
server_op.h: header della libreria server_op.a, dichiara solamente le funzioni presenti nella libreria.
lock_lib.h: definisce macro per inizializzare mutex e condition variable e per acquisire e rialsciare la lock in modo corretto secondo la politica di accesso fair per lettori e scrittori allo storage del server.
shared_queue.h: definisce le struct per l'utilizzo di code e code condivise e dichiara le funzioni implementate dalla libreria shared_queue.a per operare su tali code.
icl_hash.h: header della libreria di terze parti di Jakub Kurzak utilizzata per la creazione e la gestione della tabella hash per la memorizzazione dei file.
client.c: file contenente il main del client dove viene effettuato un controllo sugli argomenti passati a linea di comando e tramite la funzione getopt viene effettuata la scansione di tali argomenti. Le opzioni riconosciute vengono passate alla funzione execute_command, implementata nell'omonimo file.
execute_command.c: contiene la funzione execute_command che chiama opportunamente le funzioni dell'API in base all'opzione ricevuta e si occupa di stampare i messaggi (se l'opzione -p è stata specificata).
i_conn.c: file in cui è implementata L'API (dove sono presenti solo le funzioni richieste dalla specifica).
conn_supp.c: file dove sono presenti alcune funzioni di supporto necessarie per il funzionamento dell'API.
server.c: file dove si trova il main del server nel quale vengono mascherati i segnali che verranno gestiti, viene letto il file di configurazione, create le 2 piepe, inizializzati lo storage e la coda dei lavori, creati i thread a cui vengono passati come argomento degli struct che contengono puntatori alle variabili su cui devono operare, viene creato il socket e si entra in un ciclo dove con una select si aspetta che uno dei file descriptor diventi "pronto per la lettura" (i file descriptor monitorati sono: quello relativo al socket, quelli relativi alle 2 pipe e quelli relativi ai client già connessi) per poi inserirlo (se necessario) nella coda dei

lavori. Il thread main esce da questo ciclo solo se viene notificato dal thread che gestisce i segnali che un segnale di terminazione è arrivato (nel caso il segnale sia SIGHUP si smette di ascoltare il file descriptor del socket ma si continua ad ascoltare gli altri socket finché ci sono client conness). Una volta uscito dal ciclo il thread main segnala nella coda dei lavori che il segnale è arrivato ed i worker devono quindi terminare, aspetta che tutti i thread siano usciti, stampa le informazioni richieste e poi libera la memoria che era stata allocata dinamicamente.
worker_routine.c: file in cui si trova l'omonima funzione che verrà esegutia da tutti i thread worker nella quale, all'interno di un ciclo, viene letto un lavoro dalla coda, viene "decifrato" e poi eseguito con una chiamata alle funzioni della libreria server_op.a (le funzioni di questa libreria sono analoghe e simili a quelle dell'API con il quale il client interagisce col server). Se un client si disconnette il worker lo notifica al thread main mediante la pipe indicando il file descriptor del client disconnesso cambiato di segno. Se invece il worker esegue correttamente il lavoro e il thread non si è disconnesso notifica al thread main mediante la pipe di tornare in ascolto del file descriptor del client per ascoltare altre richieste. Quando il file descriptor collegato al lavoro da eseguire è uguale a -1 è il segnale che è necessario uscire, quindi il worker (dopo aver rimesso quel lavoro nella coda per far si che si propaghi agli altri thread) libera la memoria allocata dinamicamente e termina.
signal_handler.c: file contenente l'omonima funzione che verrà eseguita dal thread gestore dei segnali. In tale funzione mediante sigwait il thread si mette in attesa dei 3 segnali (che sono mascherati in quanto la maschera del thread è ereditata al thread main) da gestire e quando ne riceve uno lo notifica al thread main chiudendo il file descriptor per la scrittura sulla pipe (che causa un risveglio dalla select), modificando il valore di una variabile condivisa per indicare quale segnale è arrivato e poi termina . Sono gestiti solo i 3 segnali richiesti (SIGHUP, SIGINT, SIQUIT) quindi la ricezione di ogni altro segnale causerà l'esecuzione del gestore d default.
server_op.c: file dove sono implementate le funzioni con le quali i worker interagiscono con la memoria ed altre funzioni necessarie per il corretto funzionamento del server.
shared_queue.c:file in cui sono implementate le operazioni per la gestione di code e code condivise (utilizzate per la coda dei lavori e per le code di attesa dei file).
icl_hasc.c: file che implementa la libreria di terze parti per la gestione della tabella hash (presenti nel file i crediti all'autore Jakub Kurzak).
Le librerie del client sono compilate e linkate come librerie dinamiche in quanto si presuppone che più client siano in esecuzione in contemporanea, mentre la librerie per il server sono compilate e linkate cone librerie statiche.
Maggiori informazioni sulle funzioni sono descritte nei commenti del codice.

Gestione degli errori

Gli errori per quanto possibile vengono gestiti restituendo l'errore al chiamante e se necessario settando errno per indicare l'errore verificatosi. Molti errori sono fatali e causano quindi l'uscita del thread (nel caso l'errore si verifichi solo in un thread worker) o di tutto il processo, prima di uscire si cerca per quanto possibile di liberare la memoria allocata dinamicamente e si stampa sullo standard error l'errore riscontrato. Altri errori non fatali come la disconnessione improvvisa di un client durante la lettura o la scrittura di messaggi vengono gestiti semplicemente considerando il client come disconnesso e quindi non dovrebbero impedire il normale funzionamento del server. Il valore di errno viene utilizzato per propagare anche eventuali "errori" che si verificano nelle operazioni sui file salvati (errori descritti in precedenza nella descrizione delle operazioni) al chiamante che li convertirà in un carattere per indicare l'errore, che verrà inviato nel messaggio per il client, il quale lo riconvertirà nel corrispondente valore di errno per stampare sullo standard error (se l'opzione -p è specificata) l'errore verificatosi.