Relazione progetto

z z

September 2022

1 Scelte progettuali

Il limite di ogni file è impostato a 1024bytes, contando anche il '\0', percio ogni file puo contenere al massimo 1023bytes. Il server calcola il spazio usato trascurando '\0'.
La funzione "openFile" viene usato dal client con flags settato a "O_CREATE | O_LOCK", il server implementa i tre casi, 1. "O_CREATE", 2. "O_LOCK" 3. "O_CREATE | O_LOCK". Se il file è presente sul server, viene risposto al client con risposta "1" e il client chiama la funzione "appendToFile" per appendere input alla fine del file. Se il file non è presente sul server, viene risposto al client con risposta "0" e il client chiama la funzione "writeFile".
Il client per la strittura usa sempre open-write or append-close questa struttura, se il operazione precedente non ha avuto successo non chiama la funzione seguente.
Il client con closeConnection rilascia tutti i file lockati da lui.
Il server utilizza il numero associato al file descriptor del client per identificare il client.
Il server considera un file senza data non come un file, dato che non occupa spazio. Se durante il "writeFile" fallisce, questo file viene rimosso dal server. Se il file ha avuto "writeFile" con successo allora il file viene contato.
Viene usato il "realpath" e "basename" per il calcolo del path assoluto del file e il nome del file in caso di espulsione o lettura dal server.
Il retry time viene settato a 5s, se entro 5s il client non riesce a connettere al server viene chiuso il client.

2 Protocollo di communicazione client-server

La comunicazione è basato su stringhe di al massimo 2048byte.s

2.1 Request

Le parti del request sono separati dal ";". Ha il seguente formato "operazione;path;flags;Nread;data", se non è presente viene semplicemente saltato. Tabella di corrispondenza tra operazione e lettera:

lettera	operazione	formato di request
o	OpenFile	o;path;flags
w	writeFile	w;path;data
c	closeFile	c;path
a	appendToFile	a;path
r	readFile	r;path
R	readNFiles	R;Nread
l	lockFile	l;path
u	unlockFile	u;path
x	removeFile	x;path
C	closeConnection	C

"openConnection" connette al server e non fa il request

2.2 Answer

Il server risponde con un int al client in seguito alla richiesta,
1. caso "0", vuol dire che operazione è andato a buon fine
2. caso ">0" viene interpretato questo int in base all'oprazione, se operazione è un "readNfiles" allora è il numero di file disponibile per la lettura, se è un operazione di "writeFile", "appendToFile", il numero indica il numero di file mandato in espulsione
3. caso "<0", il client interpreta questo int con la funzione "*setErrno*" in base alla seguente tabella:

-1	errore causato da client stesso, non usato dal server
-2	EEXIST file esistente
-3	EACCESS non ha permesso
-4	ENOSPC spazio non sufficiente
-5	ENOENT file non esistente
-6	EFBIG file troppo grande

3 Parte client

Client usa la struttura dati "commandNode" per salvare le richieste ricevuta dalla linea di commando ed esegue i commandi in base all'ordine FIFO.

4 Parte server

Server utilizza la struttura dati "jobNode" per salvare le richieste ricevute dai client ed esegue le richieste in ordine in base all'ordine FIFO. E utilizza "fileNode" per salvare data, fileNode è composto da "path assoluto", "data", "clientSocketNumber", il ultimo, se è "-1" allora il file è libero, in caso diverso da "-1", il clientSocketNuber indica il client con questo numero come file descriptor ha il blocco su file
Per entrambi strutture dati sono presenti il lock, e un variabile di condizione per svegliare i thread in attesa.
Il main ha il compito di ricevere le richieste e attraverso il select, mette nella lista di "jobNode" , il worker estrae dalla lista di "jobNode" e attraverso la prima lettera individua operazione e con un switch chiama il funzione.

4.1 config.txt

config.txt ha la seguente struttura:
1. n;numeto di thread;
2. k;numero di file;
3. s;spazio di storage;

L'ordine non ha importanza, pero ";" sono necessari per identificare correttamente.

4.2 Note su lockFile

Dato che il lockFile si mette in attesa finche ottiene il lock su file. In caso di terminazione con SIGHUP, lockFile riesce a terminare con successo dato che closeConnection del detentore del lock libera tutta i suoi lock. Nel caso di SIGINT o SIGQUIT che svuota la lista di "jobNode" puo succedere che il server non riesce a terminare se proprio il ultimo thread dei worker sta eseguento il lavoro di lock e dato che nessun thread chiama signal puo risultare che il thread non viene piu risvegliato.

5 Test e log e note finali

5.1 test1

test1 ha lo scopo di testare i le varie commandi disponibili e se ci sono memory leak. output attesso:
WRITE TOTALI: 3
APPEND TOTALI: 6
WRITE BYTE TOTALE: 51
MEDIA BYTE WRITE: 5
READ TOTALI: 1

READ BYTE TOTALE: 51
MEDIA BYTE READ: 51
LOCK TOTALI: 1
UNLOCK TOTALI: 1
OPEN-LOCK TOTALI: 9
CLOSE TOTALI: 9
OUT TOTALI: 0
MAX BYTE HIT: 51
MAX FILE HIT: 3
L'ultima parte dipende dalla tid.

5.2 test2

test2 ha lo scopo di testare il funzionamento dell'algoritmo di rimpiazzamento
output attesso:
WRITE TOTALI: 12
APPEND TOTALI: 0
WRITE BYTE TOTALE: 68
MEDIA BYTE WRITE: 5
READ TOTALI: 0
READ BYTE TOTALE: 0
MEDIA BYTE READ: 0
LOCK TOTALI: 0
UNLOCK TOTALI: 0
OPEN-LOCK TOTALI: 12
CLOSE TOTALI: 12
OUT TOTALI: 2
MAX BYTE HIT: 63
MAX FILE HIT: 11
L'ultima parte dipende dalla tid.

5.3 test3

test3 ha lo scopo di testare il funzionamento del multi-threading, output varia per ogni esecuzione.

5.4 log

Il log prodotto dal server ha la seguente struttura.
numero di thread;operazione;byte di read;byte di write;path interessato;esito.
In caso di operazione non successo, read e write sono sempre messi a 0
Se il numeto di thread utilizza -1 allora vuol dire che questa operazione è un oeprazione interna del server, per esempio algoritmo di rimpiazzamento e ac-

cettazione del nuovo client.

5.5 Note Finali

I test sono stati eseguiti su una macchina virtuale con Ubuntu a due core e 8GB di ram
Il progetto si trova sulla pagina github `https://github.com/shi-zq/sol`.
Il bash puo bloccarsi per un po dato che deve leggere il log riga per riga per il calcolo dei risultati
usare make clean per eliminare i file compilati e socket
usare make test1-test2-test3 per eseguire il test. Il statistica.sh viene chiamato automaticamente dopo il test