Relazione Progetto Sistemi Operativi
Trezza Lorenzo 579726 corso A
Istruzioni per la compilazione
(cartelle necessarie per il funzionamento)
se non presenti vanno create con i seguenti comandi
mkdir libs
mkdir espulsi
mkdir letti

chmod 777 ./scripts/*.sh
make cleanall
make all

per compilare e mandare in esecuzione i test
make test1
make test2
make test3

STRUTTURE DATI AUSILIARIE

Nel file HashLFU.c vi è l'implementazione di una hashtable con liste di trabocco a probing lineare per memorizzare i file lato server; la definizione del tipo di dato “file” e la lista che viene utilizzata nella hash è implementata nell'include file “lista.h”.
È stato deciso di lavorare con un hashtable per permettere, in uno sviluppo futuro, la possibilità di aumentare la granularità, associando una lock ad ogni bucket, permettendo così un maggiore numero di operazioni eseguibili in parallelo.

L'HashMap viene gestita per permettere una politica di rimpiazzo di tipo LFU. Ogni volta che si lavora su un file viene incrementata la variabile contatore ad esso associato e viene spostato in fondo alla lista, così facendo rimane in testa alla lista il file con meno utilizzi.
Quando c'è bisogno di liberare spazio in memoria, si analizzano tutte le teste delle liste, identificando il file meno utilizzato, che verrà espulso.

SERVER

FILE DI CONFIGURAZIONE

il file di configurazione chiamato config.txt richiede che gli argomenti passati siano scritti riga per riga rispettando il seguente formalismo: ogni riga deve terminare con il carattere ';' e che il nome della variabile e il valore che si vuole associare ad essa, siano separati dal carattere '='.
È richiesto che il seguente ordine delle righe venga rispettato:

1. numero di thread
2. memoria massima
3. file massimi
4. socket name
5. log name

All'avvio il server maschera i segnali e attiva il thread che si occuperà della loro gestione; successivamente effettua il parsing del file di configurazione, memorizzando le preferenze dell'utente; e in fine attiva il pool di thread worker.
Il server, utilizzando la funzione select, determina se è in arrivo una richiesta di connessione da parte di un nuovo client, oppure è in arrivo una richiesta da elaborare da un client già connesso. Nel primo caso viene accettata la connessione e viene memorizzato il nuovo file descriptor; nel secondo caso il messaggio ricevuto viene aggiunto ad un coda messaggi.
Dopo aver ricevuto il segnale di terminazione il server si occupa di deallocare la memoria ancora in

uso, di disattivare i thread ausiliari attivati(thread workers e signal handler thread) e di stampare a schermo un sunto delle statistiche e uno snap della hashtable prima della sua deallocazione.

THREAD WORKER
lavorano secondo il problema produttore-consumatore sulla lista di messaggi; aspettano quindi che il thread main aggiunga un elemento alla lista, lo prendono in carico, determinano l'operazione da svolgere, la eseguono e rispondono opportunamente al client.

SIGNAL HANDLER
si mette in ascolto sui segnali SIGQUIT, SIGINT, SIGHUP, quando riceve un segnale setta le variabili volatili per avvisare gli altri thread attivi(incluso il thread main) quale tipo di chiusura bisogna effettuare e termina.

CLIENT

effettua il parsing della linea di comando tramite la funzione getopt, e memorizza gli argomenti ricevuti in una lista (tranne gli argomenti -f -p -h che vengono gestiti immediatamente), si itera poi sulla lista per verificare che le condizioni e gli argomenti siano rispettati.
Alla fine si eseguono le operazioni richieste dall'utente tramite la libreria condivisa API.so

API.c

Vi sono implementate le funzioni (definite nel testo del progetto) utilizzando gli include file “ops.h” e “conn.c” dove sono definite le funzioni per la comunicazione e la definizione del tipo di messaggio scambiato tra client e server.