

Scelte progettuali

Utilizzo di nomi autoesplicativi su ogni variabile, macro e funzione in modo da poter evitare commenti espliciti su determinate righe di codice (il tutto sempre per favore maggiore leggibilità possibile). In aggiunta è stato deciso di utilizzare esclusivamente, ove possibile, materiale fornito in laboratorio.

E' possibile ritrovare *connection.h* utilizzato sia dal client che dal server, in modo da garantire che entrambi utilizzino lo stesso sockname e la stessa grandezza di buffer.

Icl_hash utilizzata per memorizzare il client connessi.

Alcune macro quali *SYSCALL*, *CHECK_EQ* ed altre derivate da queste, con piccole modifiche atte esclusivamente all'eliminazione di righe di codice dovute all'esclusiva gestione di errori/stampe.

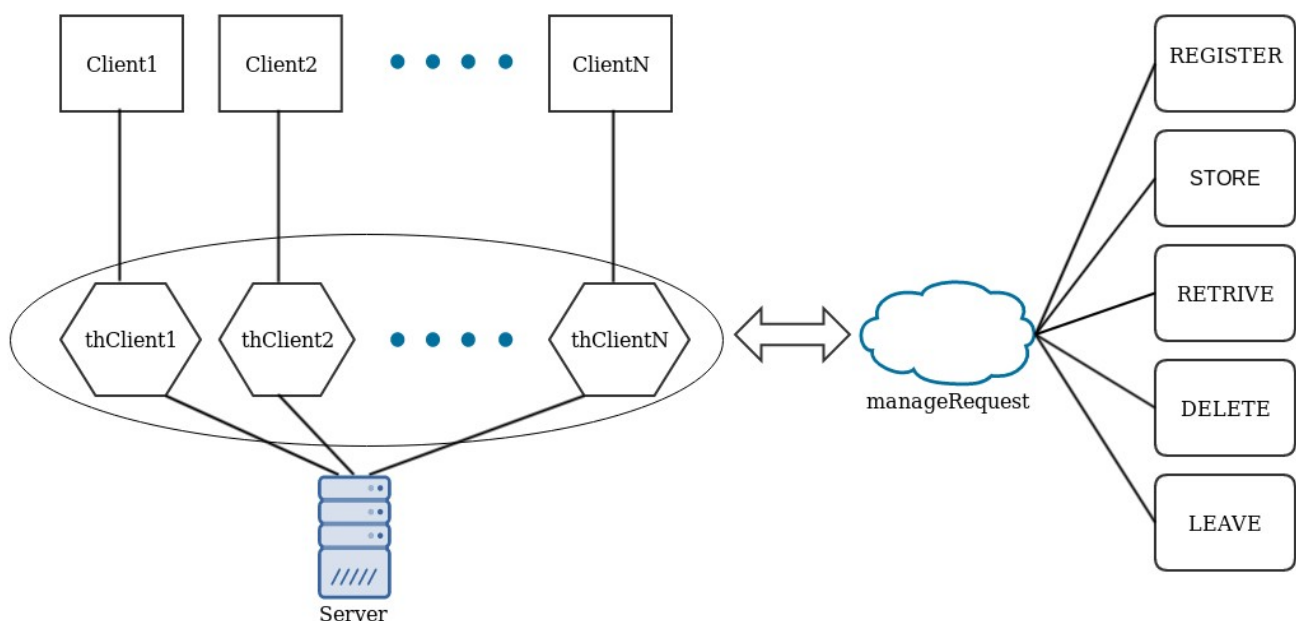
Le funzioni *countObects* e *isDot* presenti in *utils.c* sono le stesse presenti nell'es.7 file *lsdir.c*

Gestione delle richieste

Nella gestione delle richieste il server si aspetterà, nel caso in cui la richiesta non sia una STORE, un pacchetto che avrà come lunghezza massima di filename ed username 254 caratteri ognuno (più gli altri caratteri necessari per essere conformi al protocollo). Il motivo di questa scelta è relativa al numero massimo di caratteri utilizzabili in UNIX per il salvataggio dei filename. La STORE dovrà creare cartelle con l'username dell'utente registrato e filename da lui specificati, nel caso in cui arrivino grandezze non consentite dal sistema, la richiesta non andrà a buon fine.

Il server utilizza una prima read sul socket nella funzione "threadClient" (assegnata al thread che viene spawnato per ogni client connesso). Nel momento in cui riceve una pacchetto dal client, andrà nella funzione "manageRequest" che, effettuerà il parsing del comando ricevuto e lancerà la funzione corrispondente.

Diagramma esplicativo:



Gestione stato consistente dei dati

Per garantire lo stato consistente dei dati, nel caso di qualsiasi crash o terminazione, è stato impiegato l'utilizzo di un file temporaneo. Salvato nella directory `"/tmp/objStoreTmpFiles"` una volta terminata la scrittura del file, se non si verificano errori, verrà effettuato il suo spostamento tramite la systemcall `rename`.

Variabile di configurazione globali

E' possibile trovare le seguenti variabili globali realizzate per consentire un rapido cambiamento a vari parametri che potrebbero esser modificati in futuro.

- `TMPDIR`: memorizza la directory per salvare il file temporanei
- `DATADIR`: memorizza la directory dove salvare i dati
- `NBUCKETS`: numero di bucket da assegnare alla hashtable
- `SOCKNAME`: directory e nome del socket
- `MAXBACKLOG`: lunghezza massima della lista d'attesa sulla listen
- `BUFFER_SIZE`: grandezza del buffer utilizzato per la lettura
- `STARTING_SIZE`: dimensione del primo file da generare nei test
- `INC_SIZE`: dimensione da assegnare ai file successivi per raggiungere la max dimensione
- `CONTENT`: contenuto da scrivere nei file di test

Struttura dati

E' stata utilizzata la tabella hash, nello specifico l'implementazione di Jakub Kurzak (`icl_hash`) vista a lezione nell'*esercitazione n.9*.

La scelta di questa struttura dati è dovuta prevalentemente alla sua velocità generale. Come chiave hash è stato utilizzato l'username dell'utente connesso.

Concorrenza

Per la gestione della concorrenza è stata creata una variabile `mutex`, utilizzata nell'accesso alla tabella hash. Questa `mutex` è utilizzata esclusivamente nelle operazioni d'inserimento, eliminazione e ricerca.

Debugging

E' stato svolto utilizzando 2 strumenti principali.

1. Valgrind
2. AddressSanitizer (<https://github.com/google/sanitizers>)

Il primo visto a lezione, consente di eseguire un'analisi approfondita della memoria per verificare l'eventualità d'errori e/o memory leaks.

Il secondo svolge essenzialmente la stessa funzione ma con velocità di circa 20X superiori.

La decisione di utilizzare più tool differenti per l'analisi della memoria, è nata per scongiurare eventuali errori generati dai software rispetto agli effettivi errori riscontrati nella fase di sviluppo del progetto.

Nella fase di sviluppo e testing del server è stato utilizzato un file di log, contenente tutte le informazioni necessarie per agire in modo più veloce ed efficace nell'analisi degli errori.

La consegna del server non contiene le `fprintf` di debug, decisione presa a discapito di una maggiore facilità di lettura del codice.

Nello sviluppo del client sono state lasciate tutte le righe di codice utili a fornire una traccia di log del client, in quanto richiesto esplicitamente dalla specifica.

Divisione dei files

Per garantire una migliore leggibilità del codice, sono stati creati più file che contengono esclusivamente funzioni che riflettono le operazioni che ci si aspetta dal nome associato dall'header.

- **Utils:** contiene numerose macro impiegate generalmente nel checking di funzioni e la relativa gestione d'errore (solitamente la stampa di eccezioni). Sono presenti altre funzioni utilizzate sia dal client che dal server, per la creazione di pathname, filename e headers utilizzato dal protocollo di comunicazione ed altro opportunamente documentato nella classe.
- **Test:** contiene *OP_UPDATE_COUNTER* una macro utilizzata per tenere traccia degli esiti dei test. Le 3 funzioni di testing richieste dalla specifica
- **Icl_hash:** è la classe utilizzata per implementare l'hashtable già nel paragrafo "struttura dati"
- **Connection.h:** un header che contiene 3 variabili globali utilizzate sia dal client che dal server, per favorire nel caso di cambio del file del socket/buffer_size/maxbacklog una consistenza sia da parte del client che del server.
- **Lib_client:** contiene tutte le funzioni e relative implementazioni fornite dalla specifica che utilizzerà il client per effettuare le operazioni volute.
- **Objstore_server:** contiene tutto il codice che gestisce la creazione dei thread, della struttura dati e della logica del protocollo.
- **Objstore_client:** contiene il codice necessario per lanciare i test e verificare la funzionalità della libreria lib_client

Script di analisi

Lo script viene lanciato utilizzando il comando *make test*, che si assicura di pulire la cartella dei dati, avviare il server in background ed avviare il file *testsum.sh*.

Variabili globali:

- *logFile*: nome file di log.
- *clientName*: nome del programma client da lanciare.
- *serverName*: nome del programma server da lanciare.

Funzioni:

- *loopTest*: eseguire 3 cicli per lanciare in contemporanea il numero di client richiesti.
- *checkLog*: effettua il parsing di keywords specifiche, utilizzando *grep -cw*, per verificare la correttezza dei test.

Lancia il segnale *USR1* per ottenere le stats del server e successivamente invia il segnale di *SIGINT* per terminarlo.

La nota negativa di avviare il server in background, nello stesso terminale dove viene eseguito il *testsum.sh*, è nella stampa dei dati. Vedremo nella console sia l'output del server sia quella dello script. Per distinguere l'output del *testsum.sh* è stata realizzata una stampa con dei contorni e come titolo il nome dello script.

Strumenti e software utilizzati

Visual studio code: editor di testo

Git: per la gestione delle versioni

GDB: per il debugging

Valgrind e AddressSanitizer: per verificare la gestione della memoria/leaks

Make: per realizzare le regole dei file da compilare

Bash: utilizzato per generare la batteria di test

Ubuntu 19.04: sistema operativo utilizzato durante lo sviluppo

Man: per il manuale delle librerie standard

Grep: utilizzato all'interno dello script di test ed anche in fase di sviluppo per analizzare velocemente stringhe specifiche nei file di log del server