Report File Server

Progetto di Laboratorio di Sistemi Operativi

Simone Ianniciello #581201

A.A. 2020/2021

1 Server

Il server é implementato come un singolo processo multi-threaded. Ogni client é identificato attraverso il file-descriptor associato alla sua socket.

1.1 Thread principale

Esso si occupa di preparare lo stato iniziale del server per poter lavorare correttamente, oltre a creare e far partire il dispatcher e i worker threads necessari per gestire le richieste. Dopo questa fase di configurazione il thread entra in un ciclo in cui attende l'arrivo dei segnali `SIGINT, SIGQUIT, SIGHUP`.

1.2 Thread Dispatcher

Questo thread, tramite la funzione `select()`, si occupa di gestire le richieste di connessione / comunicazione dei client e di reindirizzarle, tramite una coda thread-safe, ad un worker thread libero. Non appena si riceve una richiesta da un client, il suo file-descriptor viene eliminato dal set di attesa della `select`. I worker poi notificano l'avvenuta gestione della richiesta inviando il `fd` del client attraverso una `pipe` al dispatcher che si metterà nuovamente in attesa di richieste da quel determinato client. In caso di disconnessione di un client, il worker lo notifica al dispatcher inviando `-1` attraverso la stessa pipe. La stessa `select` viene utilizzata anche per monitorare i cambiamenti di stato sulla `exit_pipe`, la quale viene chiusa nel caso in cui si debba chiudere il server (ricezione di un segnale, errore…)

1.3 Thread Worker

Non appena un worker prende in carico la richiesta di un client, effetua una lettura di un carattere dalla socket associata. Questo carattere determina il tipo di task da effettuare e di conseguenza le operazioni da eseguire per completarlo. Al termine di ogni operazione viene inviato al client un codice di errore compatibile con in valori di `ERRNO`. Le operazioni di lettura inviano anche la dimensione del file richiesto ed il contenuto.

2 Client

Il client é implementato come singolo processo single-threaded. Gli argomenti sono parsati tramite `getopt()`. Per prima cosa vengono cercati tutti gli argomenti singoli (ovvero di configurazione) cioé `-f <socket>, -p, -h`. Dopo di ché viene stabilita una connessione con il server e vengono inviate, una alla volta, le richieste dell'utente. L'interfaccia di comunicazione tra il client ed il server é data dalle funzioni presenti nella libreria `connectionAPI`

3 Configurazione del server

Il file di configurazione del server deve essere formato da una serie di relazioni <chiave>: <valore>. Le informazioni necessarie sono:

`N_WORKERS`	Indica il numero di thread eseguiti per gestire le richieste dei client
`MAX_SIZE`	Indica la dimensione massima che i file nel file server possono occupare
`MAX_FILES`	Indica in numero massimo di file che possono vivere in un dato momento sul server
`SOCKET_NAME`	Indica la posizione in cui inizializzare la socket di connessione

4 Storage

La memoria del file-server é implementata come una lista di nodi, ognuno dei quali contiene le informazioni relative ad un file. Inoltre nella struttura vengono mantenute tutte le informazioni necessarie come la dimensione / lunghezza e le variabili di condizione.

4.1 Thread Safety

L'atomicità delle operazioni sullo storage é garantita tramite l'utilizzo di due lock e due variabili di condizione. Grazie ad esse é stato implementato un sistema multi-reader / single-writer che permette di avere un solo thread in scrittura oppure molteplici thread il lettura. Ogni nodo ha anche associata una lock *'personale'* utilizzata per garantire l'atomicitá delle operazioni di `append`.

4.2 Algoritmo di rimpiazzamento

L'algoritmo di rimpiazzamento utilizza una politica `FIFO` ed é stato implementato tramite una funzione ricorsiva che elimina un file alla volta finché la dimensione o il numero di file salvati non tornano inferiori al limite fissato.

5 Gestione degli errori

La maggior parte delle funzioni sono state sviluppate in maniera da propagare gli errori a ritroso fino al chiamante che li dovrà gestire. Ad esempio se un worker richiede la creazione di un file già in memoria accadrebbe questo:

```
WORKER  →  execute(...)  →  openFile(...)  →  return EEXIST
                         write(client, EEXIST)  ←  EEXIST
```

Ovvero l'errore viene propagato fino alla funzione che invia lo stato di ritorno al client.

Se invece durante la stessa chiamata si ha un errore provando ad allocare la memoria per salvare il nome del file si avrebbe:

```
      WORKER   →   execute(...)   →   openFile(...)   →   return -1
closeAll...    ←          -1      ←          -1      ←          -1
```

In questo caso l'errore torna fino al worker, il quale decide di avviare la procedura di chiusura del processo server.

6 Compilazione e test

Per compilare entrambi gli eseguibili basterà eseguire il comando make nella cartella principale. Verranno quindi create le cartelle `build/` ed `exe/` contenenti rispettivamente i file oggetto corrispondenti ad ogni file sorgente, ed i binari del server e del client. Per eseguire i test invece si dovranno eseguire i comandi `make test[1/2]`.

Nella cartella `tests/output/test[1/2]` si troveranno i log tra cui:

client?.log Contiene l'output del client

v_client?.log Contriene l'output di valgrind relativo all'esecuzione di un client

server.log Contiene l'output del server

valgrind.log Contiene l'output di valgrind relativo all'esecuzione del server

7 Scelte d'implementazione

7.1 Comunicazione

Quando si deve spedire un file dal client al server o viceversa, dato che il destinatario non sa a priori la dimensione del file, non può allocare la memoria necessaria. Per ovviare a questo problema il mittente spedisce un messaggio formattato come: …<size>…\0<data>. In questo modo il destinatario può leggere la prima parte del messaggio (fino al terminatore di stringa) in un buffer di dimensione limitata, estrapolare la dimensione del file, ed allocare la memoria necessaria per leggere i dati.

7.2 Scrittura dei file

Quando un client richiede il salvataggio di un file in una cartella, é stato scelto di ricreare l'intero cammino a partire dalla radice in modo da evitare conflitti di nome. Ad esempio eseguendo i comandi
`-W /home/test/file.txt -r /home/test/file.txt -d /home/test/output/`,
`file.txt` si troverà in `/home/test/output/home/test/file.txt`

8 Note

Il codice é presente anche su GitHub:

`https://github.com/iannisimo/SOL_Assignment_21/`

La storico dei commit non é buono dato che ho dovuto fermare lo sviluppo per un po' di tempo e quando ho so riiniziato a lavorarci mi sono dimenticato di caricare le modifiche.