Relazione sul Progetto di Sistemi Operativi e Laboratorio

A.A. 2020/2021

Daniele Cardarelli

1 Introduzione e informazioni generali

Il progetto consiste nella scrittura di un *file storage server* in cui la memorizzazione dei file avviene in *memoria principale*.

2 Makefile

Per la compilazione del progetto vi è presente il makefile con le opzioni:

`make` L'opzione base è la regola `all`, che può essere invocata dando il comando `make`. Questa regola compila i file necessari per l'esecuzione del server.

`make clean` Per ripulire le directory dai vari file oggetto.

`make test1` La regola `make test1`, esegue il primo test descritto nella documentazione del progetto

`make test2` La regola `make test2` esegue il secondo test, che mostra l'algoritmo di rimpiazzamento del server.

3 Architettura generale del server

Il server è stato strutturato basandosi sul pattern Master-Worker, dove i thread che vengono creati all'arrivo di ogni client generano lavoro, che tramite una unbounded queue (globale e bloccante),viene preso dai workers in mutua esclusione, rimosso dalla queue e in fine risolto. Una volta che il lavoro viene risolto, ogni risposta da recapitare al client viene inserita in un'altra unbounded queue,ma questa volta (a differenza dell'altra) non è globale, ma è condivisa solo tra il worker che sta svolgendo il lavoro chiesto dal thread-client (thread che viene creato all'accettazione del client) e il thread-client stesso. I workers prendono il lavoro dalla queue globale in mutua esclusione.

Per l'accettazione dei client e l'ascolto dei segnali la scelta è ricaduta su

```
int pselect(nfds,fd_set,fd_set,fd_set,timespec,sigmask);
```

La peculiarità di **pselect** è che riesce ad attendere sia I/O su file descriptors che aspettare segnali, quindi elimina il bisogno di usare il pattern con thread e pipe per gestire i segnali attraverso select.

4 Come è stata implementata la memoria

La memoria del server è stata implementata attraverso due strutture dati:

- **Server_metadata** : la quale contiene i metadati della memoria del server e un puntatore ad una linked list di elementi di tipo *FileMemory* e

- **FileMemory** : è una struct che implementa il concetto di file su memoria principale.

Come viene realizzato FileMemory

FileMemory è implementato utilizzando la funzione della libc `FILE *open_memstream(char **bufp, size_t *sizep)` , che riesce a collegare un buffer di memoria ad un `FILE *` , questo permette di scrivere/leggere sul **file pointer** attraverso fwrite/fread e quindi risparmiare l'overhead che avremmo avuto se avessimo implementato lettura/scrittura con read/write. Stando alla pagina del manuale `man open_memstream` , ogni qual volta che viene scritto più di quanto allocato, una realloc interna viene fatta e il valore di **sizep** viene aggiornato con il valore della memoria attualmente occupata.

5 Headers

I file headers principali sono:

- `clientutils.h` : Contiene tutte le firme delle funzioni dell'API per poter parlare col server.
- `server.h` : (Principalmente) Definisce molte variabili globali utilizzate per tutto il progetto
- `utils.h` : Definisce la maggior parte dei wrapper per le funzioni "safe" e alcune macro per evitare di scrivere ogni volta l'intero nome della funzione.
- `prot.h` : Contiene al suo interno tutte le definizioni delle flag comuni utilizzate durante lo scambio di messaggi e la firma delle funzioni utilizzate per gli scambi di messaggi/
- `queue.h` Contiene al suo interno la definizione della struttura per le queue utilizzate all'interno di tutto il progetto e le funzioni per operare su di esse
- `memory.h` Definisce le strutture dati **FileMemory** e **Server_metadata**

6 File config

Nel file di configurazione ci sono presenti 6 voci:

- **[LPT]** : Path del file di log [LPT -> Log Path]
- **[SPT]** : Path del file socket [SPT -> Socket Path]
- **[NWK]** : Quantità di worker da creare all'avvio del server [NWK -> N. of workers >= 2]
- **[MFL]** : Massimo numero di file memorizzabili nel server. [MFL -> MaxFiles]
- **[MMB]** : Massimo numero di MBytes memorizzabili nel server. [MMB -> Max MBytes]
- **[PCY]** : Policy di rimozione dei file nel server, può avere solo LRU/FIFO come valori. [PCY -> Policy]

7 Protocollo e API

Il protocollo per fare richieste al server consiste nell'invio di 3 messaggi specifici da parte del client:

- Il primo messaggio è il numero dell'operazione da svolgere
- Il secondo messaggio è (solitamente) il nome del file, nel caso in cui il nome del file non è necessario una stringa temporanea viene inviata al server.

- Il terzo messaggio è nato per essere utilizzato per mandare flags, ma nel caso in cui non serva effettivamente mandare nessuna flag, lo si usa per mandare un dato in più (se necessario).

Una volta ricevuti questi messaggi il thread-client (lato server) genera del lavoro, questo viene passato ai workers, che una volta capita l'operazione da fare, andranno ad interpellare il client per ulteriori dati e/o informazioni (Nel caso di writeFile si ha che il worker chiede al client quanto deve scrivere nel file e lo stesso worker dice al client se l'operazione è compibile o meno).

Il server invece per dare l'esito al client, semplicemente invia lo status dell'operazione e nel caso sia accaduto un errore, invia anche quello. Se l'operazione è `OK` il le funzioni lato client ritornano semplicemente OK, mentre se arriva `NOT_OK` il client andrà a controllare che tipo di errore è stato inviato insieme alla risposta, e lo stampa ritornando NOT_OK.

Quando ci sono delle condizioni che potrebbero far terminare prematuramente l'operazione del server (ad esempio un tentativo di writeFile su un file che ancora nel server non esiste), il server manda al client **NO_CONTINUE** per informarlo del fatto che l'operazione non verrà continuata e poi invia lo status NOT_OK con insieme l'errore (e quindi il motivo) per cui l'operazione non è andata a buon fine, in caso contrario invia **CONTINUE**.

8 Parti di codice esterne

Sono state utilizzate le funzioni fornite dal professore dalla pagina `didawiki`

9 Test e scripts

Per testare il funzionamento del server, una volta scompattato l'archivio, basterà invocare il comando `make` per compilare tutto il progetto (è consigliabile chiamare prima un `make clean` , per evitare problemi di compatibilità). Una volta che il progetto è compilato, si può chiamare `make test1` e `make test2` .

10 Parti opzionali

Policy LRU : La policy di rimpiazzamento LRU consiste nello spostare in testa alla lista di file - il file che si ha appena cercato, cosí che i file che vengono usati di meno rimangono in fondo nella coda, permettendomi cosí di poter rimuovere il file meno utilizzato (che sará sempre quello alla coda della lista).
Logger : Il logger è una semplice vfprintf con una lock per l'accesso al file e la conseguente scrittura su file.