RELAZIONE DEL PROGETTO DEL CORSO DI LABORATORIO DI SISTEMI OPERATIVI.

ANNO 2018/2019

di William Simoni, matricola 546046

1.1 INTRODUZIONE

1.1.1 COMPONENTI DELL'ARCHITETTURA

L'architettura è costituita dalle seguenti componenti:

- **Server**. Il server è costituito da un thread in ascolto (**Main Thread**) che attende richieste di connessione tramite una **pselect** con un timer di 100 ms. Quando un client invia un messaggio di registrazione, il thread sopra citato crea un thread destinato a servire le richieste del client. Il thread servente termina quando il client invia il messaggio di deregistrazione o quando si verificano errori durante la comunicazione. Ogni thread servente esegue in loop una **pselect**, con un timer di 100 ms, per attendere un messaggio dal client, poi il **parsing** dell'header del messaggio ricevuto e infine l'operazione richiesta.
  Per evitare che due Client si connettano con lo stesso username, il server utilizza una tabella hash in cui vengono memorizzati i nomi utente di tutti i client che stanno comunicando con il server. Il server memorizza i dati inviati dal client nella directory **dati**. Il codice del server è contenuto nel file **server.c**.
- **Client.** Il client esegue a seconda dell'input uno dei tre test definiti nella specifica. In particolare, prima tenta una connessione, in caso di successo esegue i test, e termina disconnettendosi dal server. Il Client, per comunicare con il server utilizza la libreria **ObjStoreOperation.** . Il codice del client è contenuto nel file **client.c**.

1.1.2 MODULI DELL'ARCHITETTURA

Tutte le librerie sono memorizzate nella directory LIBS. Le librerie utilizzate dalle componenti, in ordine alfabetico, sono le seguenti:

- **Data**. Ha la funzione di interfacciare il server con una struttura dati. In questo modo è possibile cambiare la struttura dati senza modificare il codice del server. In particolare, **Data.c**, consente al server di eseguire operazioni sulla tabella hash e rende la tabella hash concorrente.
- **Icl_hash**. Tabella hash fornita a lezione.
- **IOop.** Contiene le operazioni che permettono al client e al server di effettuare operazioni di lettura o scrittura su un file dato il suo file descriptor. Nella libreria sono contenute le funzioni **readn** e **writen** fornite a lezione e la funzione **readst** che esegue una singola lettura di al più un numero indicato di caratteri.
- **ObjStoreOperation.** La libreria descritta nella specifica del progetto e usata dal Client per comunicare con il Server.
- **Params.** Parser per la lettura dell'header dei messaggi che si inviano Client e Server. Il Server utilizza la libreria implementata in **Params.c** mentre il Client utilizza la libreria implementata in **ParamsClient.c**. Entrambe le librerie memorizzano i vari elementi dell'header in determinate variabili di una struct denominata pars.
- **Signals.** Insieme di funzioni per settare header di segnali o maschere di segnali. È utilizzata dal Server.

1.2 DETTAGLI IMPLEMENTAZIONE

1.2.1 GESTIONE DEI THREAD

I thread serventi sono creati dal **Main Thread** in **detach mode**. Quindi quando terminano rilasciano automaticamente le risorse.
Quando il **Main Thread** riceve un segnale (diverso da SIGUSR1), prima di terminare, attende che tutti i thread attivi del processo concludano le loro operazioni. Nel dettaglio, il **Main Thread** attende che la variabile globale **numThread**, che indica il numero di Thread in esecuzione, sia uguale a zero sospendendosi sulla variabile di condizione **condNumThread**. Quando un thread servente viene creato, il **Main Thread** incrementa la variabile **numThread** mentre quando un thread servente termina la decrementa. Ogni volta che la variabile **numThread** viene decrementata, viene eseguita una signal su **condNumThread**, risvegliando eventualmente un thread sospeso su quella variabile.
Il numero massimo di thread serventi è indicato dalla macro **MAXTHREAD,** ed è posto uguale a **50**. Se viene raggiunto il massimo numero di thread serventi, il **Main Thread** si sospende sulla variabile di condizione **condnumThread**, in attesa che uno dei thread creati precedentemente termini. Dopo 200 ms di attesa, il **Main Thread** crea un thread **extra** e incrementa di 1 il numero di thread ammessi. Il massimo numero di thread ammesi sopra la soglia massima è indicato dalla macro **MAXEXTRA** mentre il numero di thread extra ammessi, dalla variabile **extraThreads**. Se una wait termina prima dei 200 ms, il numero di thread ammessi sopra la soglia massima viene decrementato di uno.

1.2.2 GESTIONE DEI SEGNALI

Una delle prime operazioni eseguite dal **Main Thread** è mascherare tutti i segnali. Tale maschera viene mantenuta per buona parte del tempo di esecuzione del **Main Thread** e dei thread serventi.
Successivamente, il **Main Thread**, modifica il signal handler array per personalizzare la gestione dei segnali:

- **SIGUSR1** è gestito dall'handler **handlerSIGUSR1** che setta **sigFlags[1]** a 1.
- **SIGINT, SIGTERM, SIGQUIT, SIGTSTP, SIGALRM, SIGSEGV** sono gestiti dall'handler **handlerOtherSignals** che setta **sigFlags[0]** a 1.
- **SIGPIPE** è ignorato.

Il Main Thread per attendere richieste di connessione dai Client usa **pselect**. La pselect, **atomicamente**, setta la maschera di default del processo, esegue la select e infine rimaschera tutti i segnali. Se durante la pselect o fra una pselect e la successiva viene inviato al processo un segnale, la select viene interrotta e il segnale gestito (se possibile). I thread serventi usano la pselect allo stesso modo del Main Thread.
Le pselect sono dotate di un timer per permettere a un thread di terminare nel caso in cui un altro thread riceva un segnale di terminazione. In assenza di un timer, il thread si bloccherebbe sulla select per un tempo che dipende dalla velocità del client di inviare una richiesta, rallentando la chiusura del Server.
Se viene ricevuto un segnale (eccetto **SIGUSR1** o **SIGPIPE**), i thread serventi, nella maggior parte dei casi, completano l'ultima operazione richiesta dal client e terminano leggendo sigFlags[0] = 1. Allo stesso modo, ache il **Main** Thread leggendo sigFlags[0] = 1 inizia la procedura di terminazione.

1.2.2.1 GESTIONE SIGUSR1

Come richiesto dalla specifica, in seguito alla ricezione del segnale SIGUSR1, il **Main Thread** stampa varie informazioni come il numero di client connessi o il numero di file o byte contenuti nella cartella **"data"**.
Questi dati sono memorizzati in variabili globali; i dati relativi alla cartella **"data"** sono inizializzati usando la funzione **nftw** e successivamente vengono aggiornati dai thread serventi in seguito a store o delete avvenute con successo; i dati relativi al numero di client e thread sono inizializzati a **0** e vengono aggiornati dai thread serventi.
Per garantire la consistenza delle informazioni ed evitare race conditions, ogni modifica a una di queste variabili è protetta da una lock. Per li stessi motivi, anche la lettura avviene solo in seguito all'acquisizione di una lock.

1.2.3 TABELLA HASH

Come già indicato, la tabella hash utilizzata è quella fornita a lezione. Per renderla concorrente è stata utilizzata la stessa lock per ogni operazione di inserimento o cancellazione. È stata presa questa decisione principalmente per non modificare il codice a discapito, molto probabilmente, di una minore efficienza.
Il server non usa direttamente le funzioni della tabella hash, ma usa l'interfaccia fornita da Data.

1.2.4 OPERAZIONI SERVER

Tutte le operazioni elencate di seguito sono eseguite dal thread servente in risposta alla ricezione di una richiesta del client servito. Per quanto riguarda la gestione della memoria, eccetto per buffer di grandi dimensioni, si è cercato di usare lo stack di ciascun thread.

1.2.4.1 *OPERAZIONE DI CONNESSIONE*

Il thread tenta di inserire lo username inviato dal Client nella tabella hash. Se l'operazione ha successo, e quindi non ci sono altri client attivi registrati con il medesimo username, allora aggiorna le variabili globali descritte precedentemente e, se ancora non esistente, crea la cartella per il client.

1.2.4.2 OPERAZIONE DI STORE

Se un file con il nome indicato dal client esiste già, allora il vecchio file con quel nome viene sovrascritto. La scrittura del file avviene utilizzando la funzione **WriteFile** che scrive, in blocchi di **2048 byte**, i dati inviati dal client nel nuovo file utilizzando le funzioni contenute in **IOop**. In caso di fallimento dell'operazione, il file viene eliminato. L'operazione modifica le variabili globali indicate precedentemente.

1.2.4.3 OPERAZIONE DI RETRIEVE

Se il file esiste viene chiamata la funzione **ReadFile**. ReadFile invia al Client il messaggio di risposta contenete la lunghezza del dato richiesto e successivamente chiama la funzione WriteFile per inviare i dati al client.

1.2.4.4 OPERAZIONE DI DELETE

Se il file esiste viene chiamata la funzione **DeleteFile**. DeleteFile elimina il file e ritorna la sua lunghezza. La lunghezza poi viene utilizzata dal thread servente per aggiornare le variabili globali descritte precedentemente.

1.2.4.5 OPERAZIONE DI DISCONNESSIONE

Il thread elimina il client dalla tabella hash e termina la comunicazione.

1.2.5 CLIENT DI TEST

Come dati di prova, il Client invia al server una serie di array di byte contenente la medesima stringa ripetuta. La stringa utilizzata è:

- ABCD4567890123456789012345678901234567890123456789W12345678901234567890123456789012345678901234567890123456789

Il Client invia 20 pacchetti di dimensione crescente da 100 a 100000 byte.

1.2.6 GUIDA ALLA COMPILAZIONE E AL TESTING

1.2.6.1 COMPILAZIONE

È sufficiente eseguire i comandi make per costruire tutti gli eseguibili. Il progetto è composto da due eseguibili chiamati **server** e **client**. Di default, il server stampa su terminale solo quando riceve il segnale SIGUSR1. Il server

può stampare ulteriori informazioni definendo la macro **DEBUG**. Per farlo è sufficiente modificare leggermente il **Makefile** aggiungendo **$(DEBUG)** al comando **gcc** che costruisce l'eseguibile del server.

```
server: server.c $(LIB)/libSignals.a $(LIB)/libParams.a $(LIB)/libData.a
    $(CC) $(DEBUG) $(CFLAGS) $(INCLUDES) $(OPTFLAGS) -o $@ $< $(LDFLAGS) $(LIBSSERVER)
```

1.2.6.2 TESTING

Per eseguire i test è necessario inizialmente avviare il server. Successivamente è sufficiente digitare **make test** che avvia lo script **test.sh** e successivamente, una volta terminato il test, avvia **testsum.sh** che analizza i dati restituiti dallo script di testing, invia il segnale SIGUSR1 al server e termina.

1.2.7 SISTEMA OPERATIVO

È stata utilizzato il sistema operativo fornito sul sito **didawiki**, ossia **Xubuntu 14.10**.