BIBLIO

Laboratorio II - A.A. 2022/2023

Gianni Pan

Matricola: 615945

Parte I

Struttura del progetto.

Il progetto è strutturato in vari file: Server.c, Client.c, format.c, format.h, queue.c, queue.h e message.h.

All'interno di Server.c e Client.c troviamo le funzioni main per avviare rispettivamente il bibserver e il bibclient, insieme a tutte funzionalità richieste dal progetto BIBLIO.

Nel file format.c è presente la logica per la formattazione dei volumi all'interno dei file_record, predisponendoli al caricamento in memoria. Il file header format.h contiene la struttura di un volume in memoria e il riferimento alle funzioni definite in format.c, in modo da poter essere inclusa nel file Server.c.

Il file queue.c, assieme al corrispondente file header queue.h, contengono la definizione della struttura condivisa nel quale sono immagazzinati i file_descriptor delle connessioni client attive e dal quale i thread worker andranno a gestire le rispettive richieste. Contiene due funzioni: enqueue e dequeue.

Il file message.h contiene le macro dedicate al tipo di richiesta/risposta, assieme a una struct contenente la struttura dei messaggi scambiati tra client/server.

La maggior parte delle funzionalità del progetto è stata implementata nel file Server.c.

Parte II

Principali Scelte di Progetto e difficoltà incontrate.

Partiamo dal discutere le scelte legate alla parte server.

Gestione del file_record.

Il caricamento in memoria dei volumi appartenenti al file_record in input è stata gestita creando una struct *Book* contenente tutti i campi possibili presenti nei vari file_record, viene eseguito poi il parsing dei record tramite un tokenise linea per linea del file per popolare la struct, utilizzando ";" come separatore di campi e il separatore ":" per recuperare separatamente campo e valore da ogni token. Con l'ausilio della funzione *trimWhiteSpace* vengono eliminati spazi vuoti in testa e in coda. I vari volumi popolati dai valori prelevati dal file_record sono poi inseriti in un array di *Book,* pronti a essere interrogati dalle query dei client.

Multithreading e Mutex.

La coda per immagazzinare le connessioni client è stata implementata nel file queue.c ed è stata sincronizzata tramite utilizzo di mutex e condition variables, definite nel file header di libreria <pthread.h> sia all'uso della enqueue sia all'uso della dequeue. Per evitare la ricreazione di thread, è presente un while loop, in cui i thread worker (consumatori) controllano all'infinito se esiste lavoro da fare all'interno della coda, se la risposta è negativa i thread aspettano passivamente, nel caso in cui viene effettuata una enqueue() a seguito di una connesione da parte di un client (Produttori), viene chiamata signal() per notificare i thread di un cambiamento. Sono state usate le condition variables per permettere una attesa passiva all'interno del while loop per un minor consumo della CPU.

Gestione del prestito.

Nel caso in cui un volume possiede un campo prestito non vuoto, per controllare se un prestito è legale, viene effettuata una operazione di conversione da stringa a time_t, struttura introdotta dalla funzione di libreria <time.h>, viene presa la data di sistema attuale attraverso la funzione di libreria *time(),* modellata per rispettare il formato all'interno dei file_record e attraverso la funzione di libreria *difftime()* viene effettuata

la comparazione tra le due date. Nel caso la differenza tra le due date è maggiore di 30 secondi, il campo prestito viene aggiornato.

Formato dei messaggi tra client e server.

La struttura dei messaggi scambiati tra client e server e viceversa, viene definita al'interno del file message.h in rispetto ai campi richiesti nelle direttive del progetto.

File di log.

Il file di log viene creato tramite fopen in modalità scrittura fornita dalla libreria <stdio.h>, in modo tale da creare il file di log se non esiste, oppure sovrascriverlo in caso esista. La struttura del file di log è conforme a quanto specificato nelle direttive del progetto.

Scelte di progetto lato client.

Controllo degli argomenti in input all'avvio del client.

Viene eseguito il parsing degli argomenti in input tramite tokenise, per controllare che tra gli argomenti esiste almeno un argomento con doppia opzione (--), vengono controllate i primi due argomenti, se nessuno dei due possiede la doppia opzione allora sollevo un errore. Per controllare l'assenza di campi duplicati tra le opzioni, utilizzo un array in cui inserisco le occorrenze incontrate. Se uno dei valori ha due occorrenze viene sollevato un errore.

Recupero dei dati di connessione.

Per recuperare indirizzo e porta di ogni server e rendere i client consapevoli di quali server sono attivi, all'interno del file bib.conf ho inserito i campi per il nome della biblioteca, l'indirizzo e porta della biblioteca e un flag per lo stato di attività della biblioteca. I client contatteranno esclusivamente solo i server con flag di attività 1. Lo stato di attività di una biblioteca verrà modificato all'attivazione e alla chiusura del server corrispondente.

Scrittura sul file bib.conf da parte di più processi server.

Per evitare l'accesso simultaneo da parte di più server al file bib.conf, nel tentativo di modificare lo stato di attività, ogni processo server una volta entrato nella sezione di apertura del file effettua un'operazione di file locking, tramite ausilio delle funzioni di libreria <fcntl.h>. In questo modo viene garantito che l'accesso al file sia eseguito da un processo server alla volta.

Gestione dei segnali.

Al captare dei segnali SIGINT e SIGTERM vengono terminati i thread in attesa tramite una condizione all'interno del while loop, dove sono bloccati i thread per notificare lo stato di chiusura del programma e premettere la loro terminazione. Poi vengono effettuate le operazioni di riscrittura del file_record aggiornato, viene effettuato il cambiamento dello stato di attività del server nel file bib.conf e infine viene chiusa la socket del server e distrutti i vari costrutti per la mutua esclusione. Infine il programma viene terminato.

Parte III

Breve descrizione delle azioni prese da server e client all'avvio dei programmi.

- Avvio i server.
- I server prendeno dal file bib.conf le informazioni di connessione relative al nome della biblioteca dato in input al programma, cambiano lo stato di attività del server attivato e si metteno in ascolto sulla porta.
- Avvio i client
- I client effettuano il parsing delle opzioni e controllano che siano legali.
- Prende le informazioni di connessione relativi ai server attivi dal file bib.conf.
- Attivano tanti thread quanti sono i server attivi e mandano le richieste.
- I server accettano le richieste di connessione e le immagazzinano all'interno di una coda sincronizzata. I thread worker dei server prendono tali richieste e le soddisfano.
- Elaborate le richieste, viene aggiornato il file di log e vengono spedite le risposte al client.
- Viene chiusa la connessione al client.
- Alla ricezione dei segnali SIGINT o SIGTERM, i server effettuano le operazioni di riscrittura e pulizia per poi terminare il programma.