Progetto Laboratorio 2

Fulgaro Niccolò
621719

Struttura del Progetto

- `bibData` è una cartella contenente il dataset delle biblioteche.
- `config` è una cartella contenente il file di configurazione `bib.conf`.
- `lib` è una cartella contenente le librerie necessarie per il progetto:
  - `bib_ds` definisce la struttura dati `BibData`, usata per registrare in RAM i record del dataset e formata da un array di stringhe, un intero che ne indica la dimensione e una mutex per accedervi. Le funzioni per lavorarci sono:
    - `createBibData`: gli viene passato il path del dataset e restituisce un tipo `BibData` contentente il dataset.
    - `recordMatch`: gli viene passato un singolo record e un tipo `Request` e restituisce `true` se il record soddisfa la richiesta, `false` altrimenti.
    - `searchRecord`: gli viene passato un tipo `BibData` e un tipo `Request` e restituisce un tipo `Response`.
    - `fileFormatCheck`: gli viene passato il path del dataset, controlla il formato, se è giusto restituisce il file descriptor del dataset, altrimenti `NULL`.
    - `loanCheck`: gli viene passato un tipo `BibData`, il numero `N` di record da controllare e l'array di interi `rec` che indica quali record controllare e restituisce `true` se non esiste il campo "*prestito*" o se il prestito è scaduto, `false` altrimenti.
    - `loanUpdate`: gli viene passato un tipo `BibData` e un tipo `Response` e aggiorna i campi "*prestito*" nel `BibData` secondo le informazioni in `Response`

    - `updateDate`: gli viene passato un tipo `struct tm *date` e un intero `days`, aggiorna la `date` aggiungendo il numero di giorni specificati in `days`
    - `requestFormatCheck`: gli viene passata una stringa contenente la `request`, un carattere indicante il `type` e il file descriptor del mittente, controlla il formato della stringa e restituisce un tipo `Request` conenente le informazioni o `NULL` se il formato è errato
    - `updateRecordFile`: gli viene passato il nome della biblioteca `name_bib`, il `path` del file di record e il tipo `BibData`, sovrascrive il file record con le informazioni aggiornate contenute in `BibData`. Se va a buon fine restituisce 1, altrimenti -1
    - `freeBib`: prende un tipo `BibData` e lo dealloca completamente
- `log_func` comprende le funzioni necessarie per lavorare con i file di log:
    - `openLogFile`: gli viene passata una stringa `name_bib` con il nome della biblioteca, apre il file di log e restituisce il file descriptor.
    - `logQuery` e `logLoan`: gli vengono passati il file descriptor del file di log, una stringa con i record da scrivere e il numero di record da scrivere.
- `pars` definisce le strutture dati `Request`, contenente due array di stringhe con i nomi dei campi e i valori da cercare, il numero di parametri da verificare, un booleano che indica se è richiesto il prestito o meno e il file descriptor di chi ha fatto la richiesta; e `Response` contenente un array di interi che indicano le posizioni dei record che soddisfano una richiesta, il numero di record che la soddisfano e il booleano per il prestito. Le funzioni per lavorarci sono:
    - `freeRequest`: dealloca completamente la `Request` che gli viene passata.

    - `requestParser`: inserisce le informazioni contenute nella stringa che gli viene passata nel tipo `Request` che restituisce, in caso di errore restituisce `NULL`.
    - `checkInputFormatNparser`: gli vengono passati i parametri inseriti da linea di comando alla chiamata del programma, ne controlla il formato e restituisce una stringa parsata, aggiornando il booleano indicante il prestito che gli viene passato. In caso di fallimento restituisce `NULL`.
    - `receive_int`: legge dalla socket, indicata dal file descriptor passatogli, un intero e lo restituisce.
    - `send_int`: invia l'intero `num` sulla socket indicata da `fd`
    - `date_extract`: converte la stringa `date` che gli viene passata in un tipo `time_t` secondo un dato formato. Restituisce la stringa convertita, in caso di errore stampa un messaggio ed esce.
  - `queue` definisce il tipo `Queue` (formato da `Node`) che rappresenta una coda di tipo FIFO, e le funzioni per lavorarci:
    - `queue_init`: inizializza la coda `q` che gli viene passata.
    - `queue_destroy`: dealloca completamente la coda `q` che gli viene passata.
    - `queue_push`: inserisce il valore `data` nella coda `q`.
    - `queue_pop`: rimuove il primo elemento della coda `q` che gli viene passata e lo restituisce.
    - `queue_is_empty`: analizza la coda `q` passatagli e restuisce 1 se è vuota, 0 altrimenti.
- `log` è la cartella in cui vengono inseriti i file di log.
- `socket` è la cartella in cui vengono create le socket.
- `bibaccess.sh` è lo script bash che elabora i file di log generati dai server
- `bibclient.c` e `bibserver.c` sono i codici principali del progetto, rispettivamente il server ed il client.
- `testing.sh` è uno script bash usato per lanciare il test dal makefile.

Test

Nello script `testing.sh` vengono memorizzati i percorsi per il server, il client e l'eseguibile "bibaccess" dai parametri che riceve. Viene verificato che tutti e tre i percorsi siano stati forniti come argomenti al momento dell'esecuzione dello script. Successivamente, vengono avviati cinque server, ciascuno con un file di testo bibliografico specifico. Dopo un secondo, vengono avviati 5 client in parallelo, ciascuno con query specifiche per accedere al database bibliografico. Questi client vengono avviati 8 volte così da mandarne 40 in totale. Dopo aver eseguito le operazioni dei client, i server vengono chiusi. Infine, viene dato il permesso di esecuzione al file bibaccess e viene lanciato due volte, ciascuna con l'opzione per effettuare una query su tutti i log dei server.

Problemi e difficoltà

Ho riscontrato la maggior parte delle nella fase iniziale di progettazione, non ho infatti avuto grossi problemi nello sviluppo dei singoli componenti del programma bensì nell'unirle in un'unica esecuzione.
L'errore stampato durante il test è `Error: invalid type ()`, viene stampato dal client quando, durante il processo di comunicazione con il server, riceve un tipo di messaggio diverso da `MSG_RECORD`, `MSG_NO` o `MSG_ERROR`, in questo caso particolare non riceve nessun carattere. Credo sia dovuto ad un errore durante la chiusura dei server e nella gestione del segnale di comunicazione.

README

I target disponibili nel make file sono i seguenti:

- `make all`: compila sia `bibserver.c` che `bibclient.c` e genera gli eseguibili
- `make server` e `make client` compilano e generano gli eseguibili rispettivamente di `bibserver.c` e `bibclient.c`
- `make server_run` e `make client_run` chiamano gli eseguibili rispettivamente del server e del client. Vengono entrambi lanciati con dei parametri di prova
- `make clean`: rimuove gli eseguibili sia del client che del server
- `make clean_sock`: rimuove il contenuto della cartella `socket`, utile in caso di errore nella rimozione delle socket da parte di `bibserver`

- `make test`: lancia 5 server e 40 client, esegue il test spiegato in precedenza
  Per lanciare direttamente da terminale le istanze del programma i comandi sono:

```
./bibserver.out name_bib dataset_path W
./bibclient.out --field="value" -p
```

dove:

- `name_bib` è il nome che si vuol dare alla biblioteca
- `dataset_path` è il path del file contenente i record
- `W` è il numero di thread worker che si vuole far eseguire al server
- `--field="value"` è il formato per inserire un parametro di ricerca, è obbligatorio inserirne almento uno
- `-p` è un parametro opzionale, se presente indica la richiesta di prestito, altrimenti si ricevono solo le informazioni relative al libro richiesto