La seguente documentazione ha come scopo quello di giustificare alcune scelte progettuali applicate durante lo sviluppo dell'applicazione distribuita Client-Server Trivia Quiz Multiplayer.

Come richiesto nelle specifiche, ogni client che invia una richiesta di connessione si aspetta di poter giocare ad un quiz senza alcuna interferenza da parte degli altri client e di poter variare la propria esperienza di gioco sulla base delle risposte che sceglie mano a mano che procede. Per questo ho ritenuto che il server concorrente fosse la scelta ottimale in modo da dedicare a ciascun client un proprio processo server e per parallelizzare i vari accessi alle domande e alle risposte presenti nel gioco.

Essendo un applicazione che potenzialmente può essere rivolta ad un numero considerevole di utenti ed avendo stabilito la necessità di un server concorrente, ho scelto di utilizzare i "processi leggeri", o thread. Infatti creare un nuovo processo server, specialmente in ambienti in cui ho un elevato numero di client, può essere costoso al contrario dei thread che sono molto più efficienti dal punto di vista delle risorse di sistema. I thread, infatti, condividono la CPU e lo stesso spazio di memoria del processo che lo contiene e questo gli garantisce anche una facile comunicazione tra loro grazie all'utilizzo di strutture dati globali e l'uso di semafori. Inoltre, il sistema potrebbe trovarsi a dover effettuare numerosi cambi di contesto tra processi server ma ,avendo i thread tutte le risorse citate sopra in comune, ho la possibilità di snellire il meccanismo e ridurre l'overhead che altrimenti potrebbe generarsi.

Per quanto riguarda la comunicazione tra processo server e processo client ho scelto di utilizzare il protocollo TCP poiché si ha a che fare con un applicazione che funziona solo ed esclusivamente se i pacchetti vengono ricevuti integri e nell'ordine corretto.

Considerando che la maggior parte dei dati sulla rete sono stringhe di lunghezza variabile (come il nickname, domande, risposte, show score, and quiz ecc), ho deciso di utilizzare un protocollo di tipo text per la trasmissione dei dati. In questo modo limito la conversione da intero a stringa ai soli interi che costituiscono una percentuale minore dei dati trasmessi e non ho l'overhead dovuto ai protocolli di tipo binario che comporterebbero la serializzazione di tutte le informazioni trasmesse sulla rete. Inoltre, avendo a che fare con stringhe di lunghezza variabile, per ogni trasmissione sulla rete, prima si invia/riceve la sua lunghezza e poi l'effettivo messaggio.

Come ultima cosa nelle specifiche è stato chiesto di gestire le terminazioni di connessione sia da parte del client che dal parte del server. Per quanto riguarda la fase di invio del messaggio in entrambi i moduli ho gestito il segnale SIGPIPE: dal punto di vista del client ho fatto in modo che, se la sessione del server dovesse venire chiusa improvvisamente, venga invocata la routine di gestione del segnale (appositamente ridefinita) che va a modificare il contenuto della macro STDERR_FILENO che rappresenta il file descriptor dello standard error in modo da stampare "Il server si è disconnesso.." e comunicare al client la fine della connessione; dal punto di vista del server ho fatto in modo che il segnale venga ignorato così che altri client possano continuare la loro sessione di gioco (se presenti) oppure che il server rimanga in attesa di richieste di connessione.

Dal punto di vista della ricezione, se si rileva la chiusura di connessione, il client la gestisce come visto a lezione, ovvero in caso di errore si ha la chiusura del socket e la terminazione del client. Nel server, oltre a quanto fatto nel client, si ha anche la terminazione del thread e l'eliminazione degli opportuni campi dalle strutture dati utilizzate per la gestione dello specifico client.

Strutture dati

La struttura dati principale utilizzata è la struct Utente. Questa si occupa di memorizzare il nickname del client che vuole registrarsi, contiene un puntatore ad un'altra struttura utente e un puntatore alla struttura Partita che si occupa di memorizzare i dati relativi alla partita che sta giocando/ha finito di giocare (tema, punteggio ecc).

Considerando il numero elevato di client che potrebbero aver inviato richieste di connessione per partecipare al quiz si è scelto di utilizzare un approccio dinamico. Ho utilizzato due liste globali (a cui si accede ovviamente in mutua esclusione con l'utilizzo dell'apposito semaforo) , rispettivamente la lista database e la lista classifica.

La prima mi è servita semplicemente di appoggio per verificare l'univocità del nickname nel momento della registrazione di un nuovo utente e per semplicità ho scelto di inserire sempre in testa. L'utente viene poi inserito nella lista classifica nel momento in cui inizia a giocare ad uno dei temi del quiz. Di conseguenza, in questa lista ciascun utente può essere presente tante volte quanti sono i temi del quiz a cui ha giocato. La lista viene sempre mantenuta ordinata (ordine decrescente) sulla base dei punteggi ottenuti durante le varie partite indipendentemente dal tema e questo mi permette di ottenere stampe ordinate semplicemente scorrendo la lista una volta e stampare in base ad un tema specifico (funzione stampaPerTema) con la complessità O(n).

Infine, si utilizza una variabile globale che mi permette di mantenere aggiornato il numero di utenti che sono connessi con il server (players). Anche per questa variabile è stato previsto un semaforo di mutua esclusione in modo da evitare che venga modificata da più processi contemporaneamente.

Moduli del progetto

Il progetto si sviluppa in tre moduli: client.c, server.c e info.h. Il modulo info.h contiene tutte le strutture dati citate sopra e lo sviluppo di alcune funzioni di utilità utilizzate poi nel modulo server.c. Sono presenti anche i file di testo domande.txt e risposte.txt da cui vengono prelevate, rispettivamente le domande per il tema scelto da inviare al client e le risposte a ciascuna domanda. Quest'ultime vengono utilizzate per confrontarle con quelle ricevute dal client e per stabilire se la risposta è corretta oppure no.