Documentazione Progetto – Gioco Trivia Quiz con Architettura Client/Server

Studente: Alessio de Dato – Matricola: 635256
Corso: Reti Informatiche
Anno Accademico: 2024/2025

1. Introduzione

Il progetto consiste nello sviluppo di un sistema di ***Trivia Quiz*** basato su architettura **client/server** con comunicazione tramite socket TCP. L'applicazione permette a più utenti di collegarsi simultaneamente a un server, scegliere un tema di domande e competere tra loro in una classifica globale aggiornata in tempo reale.

L'implementazione è stata suddivisa nei seguenti componenti principali:

- **server.c**: gestisce il caricamento dei temi, la comunicazione con i client, l'aggiornamento delle classifiche e lo stato del sistema.
- **client.c**: fornisce l'interfaccia testuale per l'utente, con gestione dei comandi, login tramite nickname e partecipazione ai quiz.
- **utility.h**: contiene le costanti comuni, le dichiarazioni di funzioni di utilità e i parametri principali (lunghezze massime dei buffer, numero di domande per quiz, ecc.).
- **compile.sh**: script di compilazione automatica per generare gli eseguibili di client e server.
- **cartella qa/**: insieme di file .txt che contengono le domande e risposte per ogni tema disponibile (cinema, fumetti, informatica, viaggi, videogiochi).

L'obiettivo progettuale era sviluppare un sistema robusto, modulare e scalabile, che garantisse correttezza della comunicazione, consistenza delle classifiche e una gestione pulita delle connessioni multiple.

2. Comunicazione Client-Server

La comunicazione è stata implementata utilizzando il protocollo TCP su socket bloccanti. Questa scelta è motivata dalla necessità di garantire affidabilità nella trasmissione dei dati e di rilevare tempestivamente la chiusura di una connessione da parte di uno degli interlocutori.

Caratteristiche principali del protocollo adottato:

- **Formato a lunghezza fissa per stringhe**: nickname, domande e risposte vengono sempre inviati con buffer di dimensione costante (definita in utility.h). Questo semplifica la logica, evitando l'invio della lunghezza del messaggio.
- **Formato binario per numeri**: il numero di categorie e l'esito delle risposte vengono trasmessi tramite tipi numerici (uint16_t), convertiti con funzioni htons/ntohs.

- **Comandi speciali**:
    - *Mostra Punteggio* → invia richiesta di classifica al server.
    - *Fine Quiz* → termina la sessione corrente e chiude la connessione.
- **Gestione delle disconnessioni**: è previsto il rilevamento immediato della chiusura del server grazie a select() (lato client) e a controlli sul ritorno di recv() e send() (lato server).

3. Struttura del Server

Il server (server.c) è stato progettato per supportare **più client concorrenti** attraverso un modello **multithread**, evitando l'uso di fork() e limitando il più possibile la complessità della gestione per i processi.

3.1 Caricamento dei temi

- All'avvio il server analizza la cartella qa/ e carica dinamicamente tutti i file .txt presenti, ognuno dei quali rappresenta un tema.
- Ogni tema è memorizzato in una struttura TemaQuiz, che contiene il nome e un array di domande/risposte (NumQuest = 5).
- Se i file non rispettano il formato previsto (coppie domanda/risposta), il server termina con un errore.

3.2 Gestione dei client

- Il server mantiene un array di slot giocatori (GiocatoreStato), ciascuno associato a un thread.
- Ogni slot contiene il nickname del client connesso e il tema attualmente in corso.
- La mutua esclusione sull'array di giocatori è garantita tramite pthread_mutex_t.

3.3 Classifiche

- Per ogni tema è definito un tabellone (Tabellone) che mantiene una lista doppiamente concatenata di record (NodoPunteggio), ordinata in base al punteggio e, in caso di parità, al tempo di completamento (finito).
- L'inserimento in classifica avviene all'inizio del quiz con punteggio 0, e la posizione viene aggiornata dinamicamente ad ogni risposta corretta.
- La sincronizzazione è garantita da un mutex dedicato per ogni classifica.

3.4 Shutdown controllato

Il server può essere terminato dall'operatore premendo **"Q/q" + Invio** sulla console. In tal caso:

- vengono chiuse tutte le connessioni attive,
- viene terminato il socket di ascolto,
- tutti i thread completano le operazioni in corso e rilasciano le risorse.

4. Struttura del Client

Il client (client.c) rappresenta l'interfaccia interattiva per l'utente e si occupa di inviare comandi e ricevere dati dal server.

4.1 Login e gestione nickname

- L'utente inserisce un nickname al primo accesso.
- Il nickname rimane memorizzato in locale (variabile g_last_nick) e può essere riutilizzato premendo semplicemente Invio.
- Il server verifica l'univocità del nickname rispetto agli utenti online.

4.2 Menu e selezione dei temi

- Il menu principale consente di avviare una sessione o uscire.
- Una volta connesso, il client riceve l'elenco dei temi disponibili.
- I temi già completati non vengono più mostrati (gestione tramite mask_done in struttura Profilo).

4.3 Gestione input

- L'input da tastiera è gestito tramite funzioni dedicate (leggiLinea e leggiLinea_reactive).
- leggiLinea_reactive utilizza select() per ascoltare contemporaneamente socket e stdin, rilevando in tempo reale eventuali spegnimenti del server.
- Comandi inline (Mostra Punteggio, Fine Quiz) sono disponibili anche durante il quiz.

4.4 Storico locale e punteggi

- Ogni sessione mantiene uno storico delle categorie completate (Completato), che viene mostrato all'utente insieme ai risultati parziali.
- Alla fine di un quiz viene mostrato un riepilogo con punteggio totale.

5. Compilazione ed Esecuzione

La compilazione è automatizzata tramite lo script compile.sh:

./compile.sh

Questo comando genera due eseguibili:

- ***./server*** – avvia il server sulla porta **4242**.
- ***./client 4242*** – avvia il client collegandosi alla porta del server.

Il sistema è stato progettato per funzionare in ambiente Linux con supporto POSIX e librerie pthread.

6. Esempio di Utilizzo

1. L'operatore avvia il server: *./server*
2. Un client si connette e inserisce il nickname *"Mario".*
3. Il server mostra l'elenco dei temi disponibili (es. *Cinema, Informatica, Fumetti*).
4. Il client seleziona un tema e riceve 5 domande in sequenza.
5. Dopo ogni risposta riceve l'esito (CORRETTA/ERRATA).
6. Al termine del quiz il punteggio viene salvato in classifica.
7. In qualsiasi momento può invocare il comando **Mostra Punteggio** per visualizzare le classifiche globali.
8. Con il comando **Fine Quiz** il client ritorna al menu principale.

7. Conclusioni

Il progetto implementa un sistema di quiz distribuito e robusto che punta a essere intuitivo ma soprattutto ben strutturato in modo da dare la miglior esperienza di gioco possibile. I principali punti di forza sono:

- **Gestione concorrente dei client** tramite multithreading.
- **Classifiche aggiornate in tempo reale** con tie-break sul tempo per classifica.
- **Interfaccia utente resiliente** rileva immediatamente lo shutdown del server.
- **Caricamento dinamico dei temi** tramite file .txt nella cartella qa/.
- **Interfaccia utente** pulita e intuitiva.

Il progetto che ho realizzato in futuro potrà essere ampliato con:

- Persistenza delle classifiche su file, per mantenerle anche tra più sessioni di server.
- Supporto per un numero variabile di domande per tema di modo da poter aumentare la complessità dei quiz.
- Realizzazione di un'interfaccia grafica o web-based.

Grazie alle scelte implementative da me fatte per sviluppare il progetto, tutte queste migliorie si potrebbero realizzare con mirati interventi, portando così il progetto, ideato e finalizzato all'esame, a diventare un progetto personale più ampio e strutturato.

Mostrando in questo modo anche un ancora più ampia multidisciplinarità tra corsi.

AdD.