Documentazione progetto Reti Informatiche

AA. 2024/2025 / Tommaso Molesti – 615820

INTRODUZIONE

Il progetto implementa il gioco "Trivia Quiz Multiplayer", un'applicazione distribuita client-server interattiva. L'obiettivo del gioco è permettere ai partecipanti di rispondere a domande inviate dal server, accumulando punti per risposte corrette e scalando le classifiche specifiche per tema. Il gioco prevede la gestione di più temi, ciascuno composto da un set di domande e risposte. Il server è responsabile della gestione delle domande, della valutazione delle risposte, dell'aggiornamento dei punteggi e del mantenimento delle classifiche per ogni tema. I client si connettono al server, scelgono un quiz a cui partecipare e interagiscono con il server inviando le loro risposte. Ci sono 4 temi disponibili: Geografia, Sport, Storia e Tech. Ogni tema è composto da 5 domande. Ogni domanda può avere al massimo 3 risposte possibili, tutte valide (non case sensitive). Ad ogni tema è associato un "gioco", cioè un'effettiva partita del giocatore a quel tema.

Le domande e le risposte si trovano nella cartella "*quiz*", nei rispettivi file "*X_domande.txt*" e "*X_risposte.txt*", dove "*X*" è un numero da 1 a 4 per identificare ogni tema. Il server carica queste informazioni in opportune strutture dati. Viene assegnato 1 punto per ogni risposta corretta, 0 se non corretta. Il server mantiene una classifica separata per ogni tema, basata sul punteggio realizzato dai giocatori. Un giocatore può giocare ad un solo tema alla volta e una volta soltanto per ogni tema.

La comunicazione tra client e server avviene tramite socket TCP, visto che è necessario trasferire dati in modo affidabile. Per lo scambio di messaggi sono state definite delle funzioni di utilità *send_message* e *recv_message*. Tali funzioni gestiscono la trasmissione della lunghezza del messaggio prima del messaggio effettivo, garantendo una lettura completa del messaggio lato ricevente. Aumenta il traffico in rete, ma semplifica la frammentazione dei messaggi. Il protocollo utilizzato per la comunicazione è di tipo testuale, in quanto le informazioni scambiate sono sempre stringhe, quindi risulta più semplice l'implementazione. Il progetto è stato strutturato in diversi moduli (file .c e .h, dentro la cartella "*utils*") per migliorare l'organizzazione del codice, la sua leggibilità e manutenibilità. Vengono gestite inoltre la chiusura del terminale sia del client che del server e l'esecuzione del comando *CTRL+C* tramite la primitiva *signal*. Quando i client/server proveranno a mandare un messaggio al server/client, in questo caso riceveranno un messaggio di errore dove sarà indicato che il server/client si è disconnesso.

La compilazione del client avviene tramite il comando *gcc -Wall client.c utils/client_utils.c -o client*, per il server invece con *gcc -Wall server.c utils/server_utils.c -o server*. Può essere anche fatta usando semplicemente il comando *make all*, che esegue il Makefile che eseguirà a sua volta questi identici comandi sopra.

L'esecuzione del server avviene con *./server*, invece quella del client con *./client* <*porta*> (di default 1234).

STRUTTURE DATI

Il progetto impiega la struttura *Player* per ogni giocatore connesso, contenente *username*, *socket* e un array di *Game* per tracciare il punteggio, lo stato (*started*, *ended*) e la domanda corrente per ciascun tema. *Theme* definisce un quiz con *name* e un array di *Question*. Ogni

Question ha un testo e un array di risposte corrette. *PlayerScoreNode* invece è una struttura ausiliaria per l'ordinamento delle classifiche prima della visualizzazione.

SERVER

E' stato scelto l'I/O Multiplexing per il server per ottimizzare l'efficienza delle risorse e semplificare la gestione della concorrenza. Questo approccio riduce l'overhead di sistema, evitando la creazione e gestione di thread per ogni client, che per un'applicazione I/O-bound come un quiz game risulterebbe eccessiva. E' la soluzione più equilibrata tra efficienza, robustezza e semplicità implementativa per le specifiche di questo progetto.

Una volta inizializzato, il server entra nel suo *select()* loop, in questo ciclo attende in modo non bloccante eventi su un insieme di descrittori di file. Questo insieme include il socket del server stesso, che viene monitorato per nuove richieste di connessione e i socket di tutti i client attualmente connessi, che vengono monitorati per capire se sono arrivati nuovi dati. Quando *select()* segnala la disponibilità di un evento, il server procede con la sua gestione. Se l'evento si verifica sul socket del server, vuol dire che c'è una nuova richiesta di connessione da parte di un client. Il server quindi accetta la connessione e ottiene un nuovo descrittore di socket per il client, quindi lo aggiunge all'insieme dei descrittori monitorati da *select()* per future comunicazioni.

Poi per ciascun giocatore il server verifica nella sua lista se ci sono dati pronti per la lettura sul rispettivo socket client. Se un client ha inviato un messaggio, il server lo riceve e lo analizza. Se il giocatore si è appena connesso e il suo username è ancora vuoto, il primo messaggio viene interpretato come il nickname del giocatore, e il server ne verifica l'unicità. Se l'username è già in uso, il server lo notifica al client e richiede un nuovo tentativo. Una volta che il giocatore ha un username valido, il server gli presenta i temi di quiz disponibili. Durante la sessione di gioco, il server valuta le risposte fornite dal client alle domande, aggiorna quindi il punteggio del giocatore e (nel caso) invia la domande successiva o un messaggio di fine quiz per quel tema. Gestisce anche comandi speciali come "*show score*", inviando la classifica attuale al client e poi riproponendo il messaggio precedente. Oppure anche "*endquiz*", che porta alla terminazione della sessione di quiz del giocatore, rimuovendolo dall'elenco dei giocatori e chiudendo il relativo socket.

CLIENT

Il client, dopo aver gestito l'eventuale parametro da riga di comando per la porta del server, stabilisce la connessione TCP con il server. Una volta connesso, il client entra in un loop principale di gioco. In ogni iterazione di questo ciclo, il client attende e riceve un messaggio dal server. Questo messaggio può essere una domanda, un menu di scelta del tema, un messaggio di stato, o qualsiasi altra comunicazione proveniente dal server. Una volta ricevuto, il messaggio viene stampato a video per informare l'utente. Dopo, il client aspetta l'input del giocatore da tastiera, che può essere una risposta ad una domanda, una scelta del tema o un comando speciale. Dopo aver acquisito l'input, il client lo invia al server per l'elaborazione.

Il client può uscire da questo ciclo e terminare la sessione di gioco in alcune situazioni. Per esempio, se il giocatore ha completato con successo tutti i quiz disponibili il server invia un messaggio di gioco finito "*finished*", provocando la chiusura della connessione e la terminazione del client. Un altro modo per terminare la sessione è che il giocatore digiti "*endquiz*", in risposta a qualsiasi messaggio del server, in questo caso il client manda il comando al server, chiude la connessione e torna al menu iniziale. Può anche digitare in qualsiasi momento "*show score*" per far si che il server gli mandi la classifica aggiornata dei punteggi di tutti i giocatori del gioco, divisi per tema.