PROGETTO DI **RETI INFORMATICHE**

635489 - FRANCESCO VESIGNA

Introduzione L'applicazione consiste in un server centrale multithreaded che permette ai client che vi si connettono di rispondere a due quiz a tema e che mantiene una classifica aggiornata dei punteggi. Ogni volta che il giocatore risponde correttamente o finisce un tema, il server provvede a aggiornare le strutture dati che codificano i punteggi di ogni giocatore per ogni tema.

Strutture Dati e Threads La struttura dati theme_array è un vettore contenente tutte le variabili necessarie per ogni tema, per essere più specifici ci sono una lista concatenata che implementa la leaderboard, domande, risposte, il nome del tema e un mutex per regolare l'accesso in mutua esclusione alla lista. Tutte le informazioni, necessarie per inizializzare correttamente questa struttura dati, sono presenti in 2 file per ogni tema. Il server appena parte si occupa di inizializzare la struttura dati facendo il parsing dei file. Per aggiungere nuovi temi basta modificare la macro NTHEMES sia nel server, sia nel client e aggiungere due file contenenti (nome, domande) e (risposte). Il server come già anticipato è concorrente e multithreaded per ridurre il più possibile il tempo per il cambio di contesto. I thread sono di due tipi, allocati staticamente all'avvio del server o allocati dinamicamente, ognuno risponde a una richiesta di connessione di un client e poi gestisce l'intera comunicazione con esso. La scelta di avere una pool di thread statici è per avere un working state efficiente, se si connette un client e non ci sono thread liberi, se ne alloca uno dinamico che alla fine della comunicazione con il client verrà distrutto, al contrario di quelli statici. Tutti i thread possiedono una struttura dati (thread_node) che è visitabile dal server e da altri thread tramite una lista concatenata (thread_list) dove ogni nodo è una struttura dati che contiene le variabili utili per la partita e una variabile condition dotata di mutex per la sincronizzazione col server. Tra le variabili per la partita c'è il nickname che è soggetto a concorrenza da parte di altri thread è perciò stato implementato un mutex apposito. Per garantire la consistenza della thread_list è implementato un ultimo mutex.

Comunicazione Il server e il client utilizzano socket bloccanti e utilizzano il protocollo TCP, in quanto l'applicazione è di tipo loss intolerant. Ogni messaggio è suddiviso in 1 o 3 parti. La prima parte contine il tipo, che codifica la azione da comunicare, la seconda e la terza contengono la lunghezza in byte del body e il body. Per tipo e lunghezza usiamo interi senza segno ceritificati (

1 e 4 byte rispettivamente per ogni architettura) entrambi vengono mandati con protocollo binary. Il body se presente è mandato in protocollo testo ed è una stringa dotata del carattere di fine stringa. Questo tipo di messaggio è particolarmente vantaggioso quando si deve comunicare solo un azione (es. l'utente richiede la classifica con showscore, il client analizza l'input e manda al server un byte che il server interpreterà come la richiesta della classifica). L'unico messaggio di grandezza possibilmente elevata è la classifica vera e propria, questo potrebbe essere soggetto a frammentazione, perciò viene eseguito un ciclo che attende/manda tutti i byte. Per messaggi più piccoli viene segnalato un errore anche se $0 < B_{RECV} < B_{TORECV}$ e $0 < B_{SENT} < B_{TOSEND}$. L'invio dei temi da parte del server è tokenizzato in modo da mandare meno byte, mentre la classifica è mandata in chiaro (massimo 10 posizioni per tema).

Client Il client si occupa di alleggerire il carico sul server facendo la validazione degli input da parte dell'utente e controllando che siano coerenti con lo stato in cui si trova. Si occupa anch'esso di tenere traccia dei temi svolti per evitare richieste al server inutili.

Server il server si occupa di stampare a video una lista aggiornata dei punteggi per ogni tema, dei temi completati e dei partecipanti. Tutte queste funzioni sono implementate prendendo il lock sulla struttura dati (sono tutte soggette a concorrenza) e scorrendola. Inoltre gestisce le richieste di connessione e le smista tra i threads.

Errori Gli errori di comunicazione (es. chiusura connessione TCP della controparte) sono analizzati dopo ogni send e recv e in base a ciò verranno codificati in una variabile (error) sia presente nel client, sia presente nel thread associato. Se viene rilevata una codifica anomala nella variabile il thread o il client salta alla fine della comunicazione, stampa il messaggio di errore e termina a meno che non si tratti di un thread statico.

Scalabilità Con un numero di connessioni TCP attive pari o minori al numero di thread statici il server risulta essere molto efficiente(non vengono creati nuovi thread e la memoria relativa alla connessione non viene ne allocata ne deallocata). Il servizio degrada in qualità quando questa soglia viene superata per via dei thread dinamici (causa allocazione e deallocazione di memoria e creazione del thread **DOPO** la richiesta di connessione da parte del client). Inoltre sarebbe opportuno $N_{temi} < 10$ per garantire una corretta visualizzazione della classifica senza causare buffer overflow.

Risposte
1) Treni, Chick Corea, Jaco Pastorius, Jazz e Rock, Lingus
2) John Isner, Novak Djokovic, Terra Rossa, Jannik Sinner, Basilea