GIORGIO CECCHI (597100) – PROGETTO RETI INFORMATICHE A.A. 24/25

L'applicazione distribuita realizza il gioco del Trivia Crack, essendo di tipo loss intolerant, ho deciso di usare esclusivamente il protocollo TCP per avere garanzie sull' arrivo (corretto) dei messaggi.

- **Gestione del numero dei temi**

Ho ipotizzato che con elevata probabilità il numero di temi a cui l'utente può giocare sia variabile nel tempo. Pertanto ho deciso di predisporre la macro NUM_TOPIC nel server, che indica il numero di temi presenti. Inoltre è presente un meccanismo tramite il quale, ad ogni connessione, il server comunica al client quanti temi sono disponibili. Domande e risposte sono salvate in due file txt distinti, i due file seguono la stessa formattazione al fine di rendere facile ed efficiente sia l'accesso in lettura durante l'esecuzione del server, sia l'eventuale aggiunta o rimozione di temi. In particolare ogni riga contiene sia il nome che le domande o risposte di un tema sfruttando il separatore ";".

In conclusione, per aggiornare l'applicazione il proprietario dovrà soltanto aggiungere o rimuovere una riga nei due file e aggiornare il valore della macro.

Un eventuale miglioramento potrebbe essere quello di rimuovere la macro e accedere al file all'avvio del server per sapere in automatico il numero di temi disponibili.

- **Formato del messaggio**

I messaggi che vengono scambiati tra i client e il server sono fortemente eterogenei, ad esempio messaggi contenti delle sigle di esito di un comando ("Error", "Ok", etc.), messaggi contenti le domande/risposte del gioco oppure, infine, messaggi contenti l'intera classifica. Ho pensato quindi che sarebbe stato altamente inefficiente definire un formato di messaggio e usare sempre quello. Questo perché sarei stato obbligato a scegliere una lunghezza considerevole e inviare quindi messaggi molto più lunghi del dovuto inutilmente. Infine ho quindi scelto di usare protocolli di tipo text e di precedere ogni invio di messaggio dall'invio di un intero contenente la lunghezza del testo. Evidenzio che per l'invio della dimensione uso il tipo certificato uint32_t e le funzioni htonl e ntohl.

Nota: è presente un'unica eccezione che rompe lo schema precedentemente descritto, ovvero l'invio del numero di temi iniziale in cui mando direttamente un intero con i tipi certificati.

Le scelte fatte per lo scambio di messaggi portano ad una criticità: quando il client chiede la classifica essa viene innanzitutto elaborata dal server, poi formattata in modo che sia già pronta per la stampa lato client ed infine inviata. Questo ha la conseguenza che, in presenza di un numero elevato di client connessi, può succedere che la classifica formattata non entri nel buffer che ha dimensione 2048 byte.

Questo caso non è gestito poiché non ho pensato ad un numero così elevato di utenti. Una possibile soluzione potrebbe essere quella di predisporre un meccanismo iterativo in cui il server manda al client la classifica divisa in più messaggi, con uno conclusivo che segnala che i messaggi che compongono la classifica sono finiti.

- **Server**

Essendo il flusso di gioco dei vari client indipendente, ho pensato che un server di tipo concorrente fosse la scelta più efficiente. In questo modo posso avere più processi, in particolare uno per ciascun client, che parallelamente ricevono richieste, le elaborano, inviano le risposte al client e infine aggiornano la dashboard del server.

Per generare la classifica è necessario andare a leggere informazioni riguardanti tutti i giocatori; ho quindi preferito usare un approccio multithread per poter sfruttare la condivisione della memoria, cosa che con approccio multiprocesso (fork()) non avrei avuto. Questa è, a mio avviso, una delle

scelte cardine del progetto per i seguenti motivi: innanzitutto ha reso facile e intuitiva la logica di esecuzione del server, poi ha permesso di non dover prevedere nessun meccanismo di comunicazione tra processi (con il risparmio del relativo overhead) ed infine porta i classici vantaggi dell'approccio multithread, quindi minore necessità di memoria e minor overhead nel cambio di contesto.

Tutti i dati che il server salva sui giocatori, nickname e punteggi, sono inseriti in una lista di strutture "player"; quest'unica struttura permette la gestione di tutte le funzionalità implementate. Essendo una lista, ed essendo molteplici i thread che vogliono accedervi, è risultato necessario gestire la mutua esclusione. Questo può essere un grosso limite per l'efficienza dell'applicazione, infatti essendo la risorsa singola ed essendo che i thread devono accedervi per qualsiasi operazione, c'è il rischio che passino molto tempo bloccati in attesa.

```
struct player {
    char client_nickname[20];
    int client_score[NUM_TOPIC];
    bool client_finish[NUM_TOPIC];
    bool in_game;
    struct player *next;
};
```

Riflettendo meglio sul fatto della mutua esclusione, mi sono accorto che per funzioni che comportano operazioni di manipolazione della lista, come l'inserimento di un nuovo nodo, è ovviamente un vincolo necessario, ed infatti è stata gestita. Invece, per tutte le funzioni che necessitano soltanto di operazioni di lettura e/o scrittura sulla lista, la mutua esclusione non è necessaria, ed infatti non è stata gestita. La motivazione che mi ha portato a questa scelta è che operazioni di questo tipo non portano al rischio di segmentation fault e che quindi la mutua esclusione, in questo contesto real time, porta soltanto a ritardi senza nessun vantaggio (si guardi il seguente esempio).

Questa scelta porta ad un grosso miglioramento in efficienza perché permette di realizzare l'operazione più onerosa, ovvero la generazione della classifica, senza che sia necessario l'accesso in mutua esclusione.

Esempio: supponiamo che contemporaneamente o quasi alcuni utenti facciano punto e altri invece chiedano la classifica.

1) *Se si gestisce la mutua esclusione i thread accederanno uno alla volta e rispettivamente segneranno il punto oppure produrranno la classifica. La classifica prodotta sarà quindi temporalmente aggiornata fino al momento in cui il thread che la genera accede, ergo non sarà aggiornata ai punteggi che i thread inseriranno subito dopo (perché ancora non hanno potuto accedervi).*
2) *Se NON si gestisce la mutua esclusione, si crea una corsa tra thread; quello che produce la classifica può ottenerla aggiornata esattamente come prima oppure parzialmente/totalmente aggiornata anche alle modifiche fatte dagli altri thread che sono o meno riusciti ad accedervi.*

In conclusione ho scelto di non gestire la muta esclusione in questo caso perché non portava nessun vantaggio.

- **Client**

Il client ha una struttura molto più semplice rispetto al server, l'unica scelta implementativa degna di nota è che ho deciso di effettuare ogni volta che è stato possibile la validazione dell'input dell'utente lato client, senza inutili invii di messaggi tra client e server.

Infine ho volutamente deciso di permettere l'utilizzo dei comandi show score e endquiz soltanto nel menu di scelta del tema e non all'interno della fase di risposta alle domande.