Documentazione

Progetto di Laboratorio di Sistemi Operativi: La partita di Tris

Fabrizio Quaranta - Gennaro Nappo
N86004300 - N86004294

INDICE

Analisi e specifica dei requisiti

Descrizione generale

L'applicazione consiste in un server multi-client per consentire agli utenti di giocare a Tris (tic-tac-toe) tra due giocatori per ogni partita, Quindi prevede funzionalità per gestire più partite simultaneamente e per sincronizzare lo stato di gioco tra i giocatori e tutti gli utenti collegati al server.

Funzionalità principali dell'applicazione

- **Accesso:** Ogni giocatore deve accedere all'applicazione indicando un proprio username.
- **Creazione di partite**: Un giocatore può creare una o più partite, ma può giocare attivamente solo a una partita alla volta.
- **Partecipazione alle partite**: Un giocatore può richiedere di partecipare a una partita;
- **Gestione delle richieste di partecipazione**: Il creatore di una partita può accettare o rifiutare la richiesta di partecipazione alla propria partita.
- **Stati della partita**: Ogni partita può trovarsi in uno dei seguenti stati:
    - **nuova_creazione**: la partita è appena stata creata, ma non ha avuto ancora una partecipazione.
    - **in_attesa**: il creatore della partita è in attesa di un'altro giocatore per iniziare la partita.
    - **in_corso**: i giocatori stanno giocando

    - **terminata**: la partita è terminata, quindi i giocatori possono scegliere se continuare a giocare.
- **Messaggi in broadcast:** tutti i giocatori connessi ricevono dei messaggi che li aggiornano sullo stato delle partite.

Motivazione delle scelte implementative

Si è deciso di assegnare ad ogni utente un unico thread durante il suo utilizzo dell'applicazione, poiché la facilità della gestione di questo tipo di parallelizzazione permette di gestire una quantità di client maggiore grazie alla leggerezza ottenuta.
Poichè l'applicazione è un gioco, il client è stato sviluppato in Java con la libreria JavaFX, in questo modo è stato possibile sviluppare un'interfaccia grafica interattiva,che garantisce un utilizzo ottimale del gioco.
Per garantire una maggiore affidabilità il numero di partite, di richieste e di giocatori sono stati limitati rispettivamente a MAX_PARTITE, MAX_QUEUE_SIZE e MAX_SOCKETS, questi però possono essere facilmente modificati nel file "config.h".

Architettura

L'architettura adoperata è basata sul paradigma client-server. Il server, implementato come un programma in linguaggio C, è responsabile della gestione completa delle partite e delle richieste di partecipazione, inoltre ha il

compito di fare da tramite tra i thread di due giocatori impegnati nella stessa partita e di inviare i messaggi in broadcast.
Ogni client è un'applicazione sviluppata in Java e rappresenta un utente che sta utilizzando l'applicazione. Il client al momento dell'accesso si connette al server, al quale invia dei messaggi in base alle operazioni effettuate dall'utente e attende delle risposte. Quando riceve un messaggio dal server, legge questo messaggio ed effettua le operazioni relative al messaggio ricevuto.

Disconnessione

In caso un utente effettui il logout, viene inviato un apposito messaggio al server con il numero di socket del client, quando il server riceve tale messaggio: elimina la socket dall array sockets; elimina dalla coda richieste le richieste relative all'utente; reinizializza le partite relative all'utente; chiude la socket.

Protocollo di comunicazione

I messaggi inviati dal client al server hanno la seguente sintassi:

Nome_modulo:Funzione:payload

- **Nome_modulo**: contiene il nome del modulo in c che contiene la funzione richiesta
    - Partita: gestisce le partite

    - Richiesta: gestisce le richieste di partecipazione alle partite
- **Funzione**: contiene il nome della funzione richiesta
    - **Richiesta.richiestaParser**: invoca la funzione del modulo Richiesta richiesta dal client
    - **Richiesta.putSendRequest**: aggiunge una nuova richiesta di partecipazione nell'array richieste e avvisa il proprietario della partita richiesta
    - **Richiesta.notificaProprietario**: invia una notifica al proprietario della partita richiesta
    - **Richiesta.deleteRifiutaRichiest**a: elimina la richiesta rifiutata dall'array richieste e avvisa al giocatore che ha effettuato tale richiesta
    - **Richiesta.putAccettaRichiesta**: accetta la richiesta e avvisa al giocatore che ha effettuato tale richiesta
    - **Richiesta.deleteEliminaRichiesteByPartitaId**: elimina tutte le richieste della partita appena iniziata
    - **FreeRichieste:** elimina le richieste relative all'utente che ha appena effettuato il logout
    - **Partita.inizializzaPartite**: inizializza tutte le partite nell'array partite
    - **Partita.partitaParser**: invoca la funzione del modulo Partita richiesta dal client
    - **Partita.getPartitaInAttesa**: invia un messaggio al client contenente tutte le partite in attesa
    - **Partita.putCreaPartita:** crea una nuova partita, aggiornando lo stato della partita e i nomi e le socket dei giocatori

    - **Partita.putMove:** modifica il campo della partita e avvisa l'avversario in base alla mossa eseguita
    - **Partita.controllaVittoria:** controlla se un giocatore della partita ha vinto
    - **Partita.controllaPareggio:** controlla se la partita è terminata in pareggio
    - **Partita.send_in_broadcast:** invia i messaggi in broadcast ai giocatori
    - **Partita.putRemach:** verifica se entrambi i giocatori voglio iniziare una nuova partita, ed eventualmente aggiorna i simboli dei giocatori, lo stato e il campo della partita
    - **Partita.eliminaRichiestaByPartitaId:** elimina la richiesta della partita conclusa
    - **FreePartite**: reinizializza le partite relative all'utente che ha appena effettuato il logout
- **Payload**: contiene i parametri da passare alla funzione richiesta

Concorrenza

Le seguenti risorse del server: partite; richieste; sockets e numero_socket possono essere accedute da più thread-client, quindi è stato necessario gestire l'accesso a tali risorse in mutua esclusione tramite l'utilizzo dei seguenti mutex: mutex_partite; mutex_richieste; mutex_sockets i quali bloccano le risorse ogni volte che un thread le accede, dopodichè le sblocca rendendole disponibili agli altri thread.