UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Triennale in Informatica

Corso di Laboratoria di Sistemi Operativi

Progetto 1: La partita di Tris

Anno Accademico 2024/2025

Salvatore Tortora
matr. N86004033

Gennaro Ventrone
matr. N86004074

Mariano Sommella
matr. N86004374

Contents

Chapter 1

Introduzione

Il presente documento descrive la progettazione e l'implementazione di un'architettura client server multi-client per il gioco del Tris, finalizzata a supportare partite simultanee in un ambiente flessibile. Il processo di sviluppo è stato gestito tramite il sistema di controllo versione Git e la piattaforma collaborativa GitHub. Le scelte tecnologiche primarie includono il linguaggio C per lo sviluppo di client e server, Docker Compose per la definizione di un ambiente di esecuzione isolato e riproducibile, e uno script dedicato per l'automazione della fase di avvio.

1.1 Struttura del progetto

tris-lso/ Client/ Comunicazione.h Dockerfile client.c funzioni.c funzioni.h strutture.h Server/ Comunicazione.h Dockerfile server.c funzioni.c funzioni.h strutture.h README.md docker-compose.yml Windows.bat LocalUnix.sh Unix.sh runWin.ps1 runmac.sh

Chapter 2

Progettazione

2.1 Requisiti Individuati

Il sistema implementa un gioco multiplayer del Tris (Tic-Tac-Toe) in architettura client-server, supportando la gestione di più partite contemporaneamente e permettendo ai giocatori di interagire dinamicamente con il sistema.
Il progetto simula un ambiente di gioco del Tris dotato di K partite simultanee, ciascuna delle quali è identificata univocamente e può ospitare al massimo 2 giocatori. All'interno di ogni partita, i giocatori sono distinti in due ruoli: il proprietario, ovvero colui che crea la partita, e l'avversario, che si unisce successivamente a una partita disponibile. È stato deciso di inserire come stati di gioco 3 opzioni, escludendo lo stato *Nuova_ Creazione* richiesta dalla traccia in quanto risultava ridondante. Gli stati di gioco sono: *Terminata, In corso, In attesa.*
All'avvio dell'applicazione, ogni client ha la possibilità di:

1. creare una nuova partita, che sarà inizialmente in attesa di un secondo giocatore,
2. unirsi a una partita esistente, selezionandola, tra quelle in stato di attesa, dalla lista aggiornata in realtime.
3. uscire dal gioco.

Una volta che una partita è completa (cioè composta da due giocatori), il gioco può

iniziare: i partecipanti si alternano nel compiere le proprie mosse, rispettando l'ordine dei turni stabilito dal server. Durante la partita, ogni mossa viene convalidata e sincronizzata tra i due client.
Terminata la partita — a seguito di una vittoria, sconfitta o pareggio — ai giocatori viene proposto un menù che consente di:

1. avviare una nuova partita (in alcuni casi specifici),
2. tornare al menù principale per scegliere nuove azioni.

Le possibilità disponibili dipendono dall'esito del match e verranno approfondite nelle sezioni successive.

2.2 Scelte architetturali

L'architettura del sistema si basa sul modello client-server. Il server gestisce interamente la logica del gioco, mentre il client mette in comunicazione il giocatore con il server: Gli utenti possono creare o unirsi a partite, disputarle, vincere, perdere o pareggiare. Al termine di una partita, ciascun giocatore è libero di decidere se continuare a giocare oppure uscire dal gioco. La gestione delle attività del sistema è implementata mediante thread multipli, con l'obiettivo di garantire un'esecuzione fluida e concorrente. In particolare, sono previsti:

- **Un thread per ogni giocatore connesso alla lobby:** Quando un client si connette al server, viene allocata una struttura dati `Giocatore` e viene avviato un thread dedicato alla lobby (tramite la funzione `threadLobby`). Questo thread gestisce le interazioni iniziali del giocatore, come la creazione o la selezione di una partita. Rimane attivo fino a quando il giocatore esce.
- **Un thread per ogni partita attiva:** Nel momento in cui due giocatori sono associati alla stessa partita, viene creato un thread dedicato per quella specifica partita (funzione `threadPartita`). Questo thread si occupa della comunicazione

tra i due giocatori, della gestione dei turni, della validazione delle mosse e della verifica delle condizioni di fine partita.

- Non viene invece creato un thread indipendente per ciascun giocatore durante lo svolgimento della partita: i giocatori sono gestiti tramite i thread della lobby o della partita, a seconda dello stato in cui si trovano.

L'uso dei thread, in combinazione con meccanismi di sincronizzazione come i mutex, consente di gestire in modo sicuro e scalabile più partite e giocatori contemporaneamente, evitando condizioni di race e conflitti nell'accesso alle strutture condivise, come la lobby.

2.3 Componenti principali

Abbiamo quindi identificato le tre componenti principali del sistema: i giocatori, le partite e la lobby. Nel codice, ciascuna di queste entità è rappresentata da una specifica struttura dati, che riportiamo di seguito.

Struct Giocatore

```
typedef struct {
    int socket;
    int stato;
    int id;
} Giocatore;
```

La struttura Giocatore rappresenta un client connesso al server. Essa contiene il file descriptor del socket associato al giocatore, utilizzato per tutte le operazioni di comunicazione con il client e lo stato, utilizzato per verificare se il giocatore sta scegliendo una partita.

Struct Partita

```
typedef struct {
    char nomePartita[20];
    Giocatore giocatoreAdmin;
    Giocatore giocatoreGuest;
    int Vincitore;
    char Griglia[3][3];
    int statoPartita;
} Partita;
```

All'interno della struct Partita sono memorizzate le informazioni essenziali per gestire la partita, tra cui: i due giocatori (admin e avversario), la socket del vincitore, utilizzata per notificare l'esito, la griglia di gioco, lo stato della partita (in attesa, in corso o terminata).

Struct Lobby

```
typedef struct {
    Partita partita[MAX_GAMES];
    pthread_mutex_t lobbyMutex;
} Lobby;
```

Infine in struct Lobby oltre l'array delle partite, include anche un mutex (lobbyMutex), utilizzato per garantire l'accesso esclusivo alla struttura nei casi in cui più thread tentino di accedervi o modificarla contemporaneamente. È importante specificare che la struttura è definita come struttura globale, in modo che ogni giocatore può visualizzare le partite disponibili.

Struct Giocatori

```
typedef struct {
    Giocatore giocatore[MAX_CLIENTS];
} Giocatori;
```

Infine la struct Giocatori è una struct ausiliaria utilizzata per tenere traccia dei giocatori connessi

Chapter 3

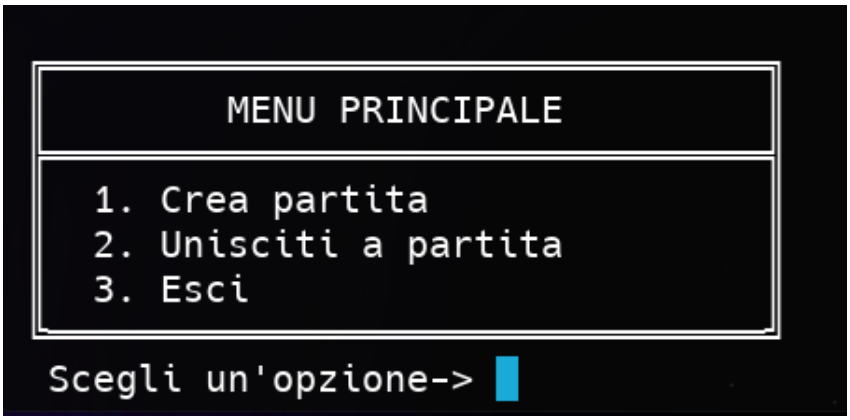
Implementazione

3.1 Comunicazione Client-Server

La comunicazione tra client e server avviene tramite **socket TCP**, attraverso le quali il server ascolta le connessioni in arrivo sulla porta definita. Per facilitare un riavvio rapido del server senza errori di binding, viene abilitata l'opzione **SO_REUSEADDR**. Ogni nuova connessione accettata dà origine alla creazione di un thread dedicato, gestito dalla funzione threadLobby, che utilizza una struttura Giocatore per mantenere le informazioni relative alla connessione, in particolare la socket associata al client.
La comunicazione vera e propria si basa sull'**invio** e la **ricezione di messaggi** attraverso i socket, seguendo un protocollo definito nel file **Comunicazione.h**, che raccoglie tutti i comandi e le risposte previste dal sistema. I thread della lobby sono responsabili della gestione iniziale del giocatore, dall'interazione con il menu fino alla creazione o unione a una partita. Quando due giocatori si uniscono a una stessa partita, viene invece avviato un nuovo thread, threadPartita, incaricato di gestire l'intera sessione di gioco e la comunicazione tra i due partecipanti.

3.2 Funzionalità e comportameti di una partita

All'avvio dell'applicazione, il giocatore si connette al server e riceve un menu principale:



Se sceglie di creare una partita, viene inizializzata una nuova struttura di gioco e il giocatore assume il ruolo di admin, in attesa di un avversario. Se invece decide di unirsi, può selezionare tra le partite disponibili identificato come Guest, una volta entrato, la partita ha inizio. Durante la partita, i due giocatori si alternano a turno, con l'admin che muove per primo. Dopo ogni mossa valida, il server aggiorna la griglia e verifica la presenza di un vincitore.

Al termine della partita si hanno 3 casi:

- **Vittoria**, il giocatore vincente ha la possibilità di scegliere se giocare ancora: se lo fa, viene creata una nuova partita in attesa di un nuovo avversario.
- **Sconfitta**, il giocatore sconfitto, viene riportato automaticamente al menu principale come richiesto.
- **Pareggio**, entrambi i giocatori vedono il menu con la possibilità di giocare ancora o uscire: la partita può ripartire solo se entrambi selezionano l'opzione per rigiocare. Se uno dei due rifiuta o si disconnette, la partita termina e si ritorna al menu principale.

Chapter 4

Docker

Il file docker-compose.yml, situato nella root del progetto, definisce due servizi: server e client, specificando per ciascuno la directory dove si trova il rispettivo Dockerfile. Il servizio client è configurato per avviarsi solo dopo il server (depends_on) e ha l'input da terminale abilitato (stdin_open, tty) per poter interagire con l'utente. L'opzione network_mode: "container:server" consente al client di condividere lo stesso stack di rete del server, permettendo la comunicazione diretta tra i due container tramite 127.0.0.1, senza esporre porte sulla macchina host.