BATTAGLIA NAVALE MULTIUTENTE

Tesina per il corso di Sistemi Operativi A.A. 2023/2024

Autori: Leonardo Marasca 0307861, Emanuele Italiani 0310987

SPECIFICHE DEL PROGETTO:

Realizzazione di una versione elettronica del famoso gioco "battaglia navale" con un numero di giocatori arbitrario. In questa versione piu' processi client (residenti in generale su macchine diverse) sono l'interfaccia tra i giocatori e il server (residente in generale su una macchina separata dai client). Un client, una volta abilitato dal server, accetta come input una mossa, la trasmette al server, e riceve la risposta dal server. In questa versione della battaglia navale una mossa consiste oltre alle due coordinate anche nell'identificativo del giocatore contro cui si vuole far fuoco. Il server a sua volta quando riceve una mossa, comunica ai client se qualcuno e' stato colpito se uno dei giocatori e' il vincitore (o se e' stato eliminato), altrimenti abilita il prossimo client a spedire una mossa. La generazione della posizione delle navi per ogni client e' lasciata alla discrezione dello studente.

INTRODUZIONE AL PROGETTO ED UTILIZZO:

L’obiettivo del progetto è di permettere a più utenti di giocare partite di una versione di battaglia navale contemporaneamente. Ogni partita viene giocata da 3 giocatori e viene gestita dallo stesso.

In questa versione di battaglia navale, l’utente che vuole iniziare una partita si connette al server, il quale attente che si raggiunga la soglia di 3 giocatori prima di iniziare una nuova partita. A questo punto verrà chiesto agli utenti di scegliere un username con il quale identificarsi e una volta che tutti lo avranno inserito, il server abilita il primo utente ad essersi connesso ad inserire la mossa per colpire l’avversario scelto. Ogni giocatore ha una quantità di tempo pre-definita per l’inserimento della mossa, allo scadere del timer, il giocatore corrente verrà eliminato e di conseguenza il gioco si chiuderà per tutti i giocatori, col server che resterà attivo ed in ascolto per nuove connessioni.

La gestione del timer per l’inserimento di una mossa viene effettuata tramite una **select()** presente all’istante della richiesta di input da parte del client.

La mossa consiste in 3 cifre nel formato “X Y Z” dove X e Y sono le coordinate della mossa e Z è l’id del giocatore obiettivo da colpire.

Non è richiesto al giocatore di disporre le navi sulla griglia in quanto tutte le navi di questa versione sono di dimensione singola e vengono generate casualmente dal server all’inizio della partita. La generazione delle griglie contenenti le navi viene gestita con la chiamata in **game_handler** delle funzioni **initialize_grid(grid)** e **place_ships(grid, ships**). Lo stesso server le stamperà subito dopo la creazione grazie alla **funzione print_grid(grid, title)**.

Ogni giocatore può visualizzare le griglie degli avversari mascherate per non rivelare la posizione delle navi ma aggiornate ad ogni turno in base all’esito della mossa corrente effettuata. Ciò viene gestito grazie alle funzioni **initialize_grids()**, **print_grid(grid, title)**, e **update_grid(grid, x, y, result)**.

L’esito di ogni mossa e l’obiettivo vengono comunicati ad ogni turno anche agli altri due giocatori che sono in attesa del turno, in modo da replicare la situazione reale del “faccia a faccia” in cui ogni giocatore sa cosa accade in tempo reale.

Quando il primo giocatore viene eliminato ad esso viene chiesto se intendo giocare una nuova partita, il server comunica la sua eliminazione ai due restanti, che passano alla modalità finale di gioco 1vs1, visualizzando nuovamente le rispettive mappe.

Ora i giocatori continueranno ad alternarsi nel fare le mosse inserendo solo le coordinate “X Y” fino a che tutte le navi di uno dei due non saranno colpite e quindi sarà dichiarato il vincitore. A questo punto il gioco finisce e chiede ai giocatori se vogliono iniziare una nuova partita.

DISCUSSIONE DELLA STRUTTURA, SCELTE TECNICHE E DETTAGLI:

L’architettura client-server utilizza un protocollo TCP per comunicare, sulle apposite socket, con i comandi send() e receive() controllando costantemente la presenza di errori di comunicazioni o interruzioni di connessione tramite le funzioni **send_data**, **send_int**, **receive_data**.

- ***Lato server:***

Il server gestisce le partite generando un **nuovo thread** ad ogni nuova partita a cui è associata una **struct game** apposita creata, e la cui memoria occupata a fine partita viene liberata con **free(game)**, e richiama la funzione **game_handler**, nella quale vengono successivamente generati 3 thread, associati alle funzioni **currentHandler, OppHandler3, Opp2Handler3**, ognuno associato alla **struct data** creata da game_handler per la partita e che gestiscono la comunicazione con un ruolo dei giocatori. Successivamente all’ingresso nella modalità di gioco 1vs1, si entra nella funzione **two_players** la quale crea un nuova **struct data2**, e due nuovi threads legati alle funzioni **current2handler** e **opp2handler** per gestire la comunicazione con i ruoli di current_player e target.

La funzione main del server è incaricata di accettare e configurare le connessioni da parte dei client che vogliono iniziare una partita, eseguendo ciclicamente queste funzioni così da garantire la possibilità di giocare più partite contemporaneamente.

Alla creazione di ogni struct viene eseguita una **malloc** per allocare la memoria necessaria al loro utilizzo, memoria ottimizzata con la liberazione tramite free(struct_name) nei momenti di chiusura di una partita. Lo stesso vale per ogni thread figlio creato, al momento della fine del codice, si esce dal thread chiudendolo esplicitamente, insieme al socket e alla struttura dati utilizzata.

Per iniziare, bisogna avviare un server con il comando “ file.c 8080 127.0.0.1 ”. Avviato il server crea una **socket**, effettua il **binding** porta-indirizzo ed accetta le connessioni da parte dei client tramite le funzioni **initSocket(char *port, char *ipAddress), ipcontrol(char *ip), portvalidate(char *pstring)** in grado di rilevare immediatamente la presenza di errori.

Dopo aver accettato la connessione di 3 clients, le nuove partite vengono iniziate secondo uno schema **Round-Robin**, in cui circolarmente a code **FCFS** di 3 connessioni, il nuovo thread che gestirà la partita con la funzione game_handler e la struttura new_game, verrà generato. Le informazioni riguardanti la connessione degli utenti vengono salvate nelle strutture e passati come argomento alle funzioni e threads in carico di gestire quella determinata partita.

Secondo la logica delle **code FIFO** inoltre, il primo giocatore ad aver richiesto la connessione al server inizierà a scegliere la mossa per primo, indifferentemente dall’ordine di invio degli username richiesto per poter iniziare.

Le attività dei threads, e la comunicazione con i client connessi, viene sincronizzata utilizzando dei semafori, dichiarando **struct sembuf oper** ed assegnando 4 semafori alla gestione della modalità di gioco a 3, di cui uno usato da game_handler parte impostato ad 1 e gli altri 3, 1 per funzione che gestisce un ruolo nella partita, a 0. Quando si passa alla versione a 2 giocatori, i semafori creati in precedenza per quella partita vengono eliminati per evitare errori di concorrenza e spreco di CPU

e memoria e si creano i 3 semafori che gestiranno il gioco 1vs1 secondo la stessa logica usata nel gioco a 3.

La gestione dei segnali viene effettuata per: **SIGINT, SIGTERM, SIGPIPE**. I primi due sono gestiti con le funzioni **funcsignal(), offsignal() ed offgame().** Per gli errori derivanti da SIGPIPE invece, si ignorando tramite **signal(SIGPIPE, SIG_IGN)** per evitare il crash del server ma ogni errore nella comunicazione, sia lato invio che di ricezione, sia server che client, viene meticolosamente gestito grazie ai controlli presenti nelle funzioni **send_data**, **send_int**, **receive_data**. Esse sono in grado di far terminare fluidamente il gioco in base alla situazione, che si tratti di chiusure lato client o server.

- ***Lato client:***

Per iniziare, bisogna avviare il client con il comando " file.c 8080 127.0.0.1 ". Avviato il client crea una **socket**, effettua il **binding** porta-indirizzo e si connette al server tramite le funzioni **initSocket(char *port ,char *ipAddress)**, **ipcontrol(char *ip)** e **portvalidate(char *pstring)** in grado anche di rilevare immediatamente la presenza di errori.

Effettuata la connessione, si attente l'inizio della partita da parte del server quando si raggiunge la soglia dei 3 giocatori e si creerà il thread che gestisce la partita tramite **game_handler.** Iniziata la partità sarà richiesto l'inserimento di un username che identificherà il giocatore durante la partita corrente, quando tutti i giocatori avranno fatto la loro scelta, i loro username saranno visualizzati, associati ad un ID con cui colpirli e alla loro griglia che verrà aggiornata durante il gioco in base agli esiti delle mosse di tutti i giocatori. Per far ciò sono chiamate le funzioni **initialize_grids()** e **print_grid(int grid[GRID_SIZE][GRID_SIZE], const char *title).**

Dopodichè si attende il messaggio dal server che abilità all'inserimento mossa. Se è il momento di giocarla, la si inserisce ed invia, nel rispetto di un timer di 30 secondi che ci disconnetterà se si impiega troppo tempo a giocare la mossa. La mossa viene inoltrata al server che determinerà l'esito, aggiorna le griglie, invia l'esito e il client visualizzerà la nuova griglia ed il messaggio di esito grazie a **update_grid(opponent_grid,X,Y,Z)**;.

Un secondo timer è pronto a disconnettere l'utente dalla partita se il gioco non progredisce per più di 2 minuti di tempo a causa degli altri utenti connessi.

Se la mossa non è valida perché non rispetta il formato, coordinate, cella già usata, o limiti della mappa, verrà visualizzato un messaggio e richiesto un nuovo inserimento.

Se è invece il momento di attendere il proprio turno, si riceverà la notifica dell'esito della mossa fatta da chi sta giocando e la griglia del target sarà aggiornata e visualizzata allo stesso modo di chi ha giocato la mossa.

Ogni volta che un giocatore viene colpito, si effettua il conteggio delle navi rimanenti e se è stato eliminato i due giocatori ancora attivi vengono mandati nella modalità **twoplayers.**

Ogni volta che una partita termina per eliminazione, vittoria, disconnessioni o errori di connessione, tutti i giocatori precedentemente connessi alla partita hanno la possibilità di cercare una nuova partita inserendo il tasto "Y" in input al momento della richiesta. Scegliendo di giocare nuovamente il codice effettuerà una chiamata ad **execv().**

Se si decide di non giocare nuovamente, la connessione verrà chiusa. La gestione dei segnali di SIGINT, SIGTERM, e SIGPIPE viene fatta in modo analogo al server.

Funzioni, strutture e simbologia:

Simbologia:

Le navi sono rappresentate su una mappa come:

- 0: Nessuna nave.
- 1: Una nave.
- 3: Nave colpita.
- 2: Cella vuota colpita.
- ~ : Maschera per i giocatori

Finché una cella non viene colpita essa sarà mostrata come “~”, quando una nave viene colpita il valore della cella nella sua posizione passa a 3 e il giocatore leggerà “X”, nel caso di una cella vuota colpita, si passa a 2 e visualizzerà “M”.

Strutture Dati

Le principali strutture dati utilizzate nel codice servono a organizzare e memorizzare le informazioni di gioco, facilitando la gestione della partita da parte del server.

- La struct Game è la principale struttura dati utilizzata dal server. Essa contiene i file descriptor dei socket relativi ai client connessi, permettendo di identificare le connessioni attive. Oltre ai file descriptor, la struttura memorizza le tre griglie di gioco, una per ogni giocatore, e il numero di navi di ciascuno. Un altro elemento chiave contenuto nella struct Game è l'ID della partita, utile per identificare univocamente ogni sessione di gioco. Questa struttura viene creata all'interno della funzione main e popolata con i file descriptor ottenuti tramite accept() quando i client si connettono al server.

- Un'altra struttura dati fondamentale è struct Data, utilizzata per la gestione della partita a tre giocatori. Questa struttura viene passata come argomento ai thread, che la utilizzano per ricevere informazioni sulla partita. Essa contiene i file descriptor dei socket, la griglia dell'avversario, i descrittori dei semafori necessari per la sincronizzazione del gioco, un flag denominato terminate_flag che segnala la fine della partita a tre giocatori, le coordinate della mossa corrente (x e y) e un array coordinates che serve per informare gli altri giocatori sulle coordinate colpite. Infine, include la variabile target_fd, che rappresenta il file descriptor del giocatore colpito.

- Per la gestione della modalità uno contro uno, viene utilizzata una versione semplificata della struttura Data, chiamata Data2. Questa struttura ha la stessa funzione della precedente, ma è adattata alla modalità a due giocatori, rimuovendo le informazioni relative al terzo partecipante.

Funzioni Principali del Server

- La funzione game_handler(void *game_data) è responsabile della gestione completa della partita a tre giocatori. Al suo interno vengono inizializzate le griglie di gioco e posizionate le navi. Il gioco viene gestito con un sistema a turni basato su un algoritmo ciclico di tipo Round Robin, in cui ogni giocatore ottiene il controllo per un turno specifico. La funzione gestisce inoltre la sincronizzazione tra i giocatori attraverso quattro semafori, i quali assicurano che i client ricevano i dati nella giusta sequenza. Durante il gioco, i nomi dei giocatori vengono scambiati tramite send() e recv(), e vengono continuamente aggiornate le griglie per mostrare lo stato delle navi colpite.

- Quando uno dei tre giocatori viene eliminato, la funzione twoplayers() viene invocata per gestire la transizione alla modalità uno contro uno. Questa funzione si occupa di creare i nuovi thread per i due giocatori rimanenti e di reimpostare i semafori per garantire il corretto svolgimento del gioco a due. Il turno di gioco viene gestito in modo simile alla modalità a tre giocatori, alternando i partecipanti e aggiornando le griglie di gioco in base alle mosse effettuate.

- La funzione creaThreads(Data *new_data) viene utilizzata per la creazione dei thread della partita a tre giocatori. Ogni giocatore ha un proprio thread associato, il quale è responsabile della gestione delle sue mosse e della ricezione dei messaggi dal server.

- I thread dei giocatori sono gestiti attraverso le funzioni current_handler(), OppHandler3() e Opp2Handler3(). La prima funzione gestisce il giocatore che ha il turno di gioco, inviando una richiesta al client per ricevere le coordinate della mossa e inviando successivamente il risultato dell'attacco. OppHandler3() si occupa invece del secondo giocatore, che riceve un messaggio di attesa mentre il primo effettua la sua mossa e, se necessario, l'esito dell'attacco se viene colpito. Opp2Handler3() svolge un ruolo analogo per il terzo giocatore.

- Nella modalità a due giocatori, le funzioni current2handler() e opp2handler() gestiscono rispettivamente il primo e il secondo partecipante. current2handler() invia al client il messaggio che richiede l'inserimento delle coordinate per la mossa successiva, mentre opp2handler() si occupa del giocatore avversario, il quale attende l'esito dell'attacco.

Sincronizzazione e Comunicazione

- Per la comunicazione tra server e client, vengono utilizzate le funzioni send_data() e receive_data(), che permettono lo scambio di informazioni in modo sicuro. La funzione send_int() viene utilizzata per inviare valori interi, mentre check_elimination() verifica se un giocatore è stato eliminato monitorando il numero di navi ancora in gioco.

- Il server include anche funzioni di gestione della chiusura e della gestione dei segnali. La funzione offgame() si occupa di chiudere il socket del server, mentre offsignal() viene invocata in caso di ricezione del segnale SIGINT, assicurando una chiusura sicura della partita. funcsignal() gestisce i segnali SIGINT e SIGTERM, permettendo di terminare il server in modo controllato.

Funzioni Principali del Client

- La funzione initialize_grids() si occupa di inizializzare la griglia di gioco, impostando ogni cella a uno stato predefinito. print_grid() viene utilizzata per mostrare la griglia a schermo, mentre update_grid() aggiorna il contenuto della griglia in base all'esito della mossa effettuata.

- Per la gestione della comunicazione con il server, le funzioni send_data() e receive_data() vengono utilizzate per inviare e ricevere le mosse, mentre twoplayers() gestisce la modalità a due giocatori. In particolare, twoplayers() utilizza la funzione select() per impostare un timer di 30 secondi, garantendo che un giocatore non possa rimanere inattivo troppo a lungo, penalizzando così l'avversario. Se un giocatore non risponde entro il tempo limite, viene considerato come abbandono della partita.

- Altre funzioni importanti includono initgame(), che riceve l'ID della partita e i nomi degli avversari dal server, e game_handler(), che coordina la partita a tre giocatori dal lato client. Infine, play_again() permette ai giocatori di decidere se rigiocare o abbandonare il gioco

dopo la fine della partita, mentre client_handler() gestisce la chiusura del client in caso di segnale SIGINT.