Relazione elaborato
Sistemi Operativi

Zeno
Dimitri

A.A. 2023/2024

Indice

Struttura e File

ipclib.h: Questo file header contiene la definizione di semun, struttura per l'inizializzazione dei semafori, la struttura Game (porzione di memoria condivisa atta a permettere il gioco tra i diversi processi) e la dichiarazione delle funzioni poi definite "create_shm", "attach_shm", "semOp", "semCtl".

printable.h: Questo file header contiene la definizione di diverse costanti utili per accorciare e rendere più chiara la sintassi per la colorazione e formattazione del testo nel terminale.
Vengono inoltre dichiarate quattro funzioni (approfondite nel corrispondente file c) "printVictory", "printTable", "*printTimer", "waitingForPlayers".

ipclib.c: In questo file sono definite le 4 funzioni precedentemente dichiarate in ipclib.h:

- (int) create_shm(): si occupa di creare una key valida e grazie a quest'ultima viene creata una porzione di memoria condivisa il cui ID viene ritornato dalla funzione.

- (Game*) attach_shm(): si occupa di generare una key (la medesima creata da create_shm, grazie alla funzione ftok() acquisisce l'ID della memoria condivisa con quella determinata key e ritorna la porzione di memoria condivisa "agganciata".

- (void) semOp() e semCtl(): funzioni per la semplificazione delle operazioni sui semafori come incremento, decremento, eliminazione etc...

printable.c: In questo file sono definite le 4 funzioni precedentemente dichiarate in printable.h:

- (void) printVictory(): si occupa di visualizzare la schermata di vittoria, pareggio o perdita.

- (void) printTable(): stampa a video la tabella del tris, con i corrispondenti simboli dei giocatori acquisiti grazie alla struttura game, (che ricordiamo risiedere nella memoria condivisa).

- (void*) printTimer(): stampa a video il tempo rimanente prima che il turno venga ceduto all'avversario. L'implementazione della funzione gestisce anche lo spostamento del cursore, in modo da garantire un conto alla rovescia su di una sola riga. Nel caso il timer specificato dal player sia 0, allora attende finché il giocatore che detiene il turno non ha compiuto la sua mossa.

- (void*) waitingForPlayers(): stampa a video lo status di join della partita dei player uno e due. Nel caso il player non sia ancora entrato verrà visualizzata una croce rossa, altrimenti una spunta verde.

- Sono poi presenti altre funzioni che gestiscono i movimenti del cursore e la pulizia della linea/schermo.

Funzionamento generale

TriServer

Per prima cosa il server controlla la correttezza degli argomenti forniti, in caso di argomenti non validi il server termina con un messaggio di errore che riporta il corretto uso del programma

Il server controlla poi se altri processi dello stesso tipo (TriServer) sono già in esecuzione, in caso affermativo il server termina e avvisa l'utente che un altro server è già in esecuzione; questo per evitare conflitti con la memoria condivisa di altri server che usano la stessa chiave e per non interrompere altre partite già in corso.

Successivamente vengono creati ed inizializzati semafori e memoria condivisa. In caso un segmento di memoria condivisa con la stessa chiave sia già presente, si assume che questa sia stata creata da una precedente esecuzione del server terminata in maniera anomala, questa viene cancellata e viene creato e inizializzato un nuovo segmento di memoria condivisa. Infine vengono gestiti i segnali e il server è pronto ad accogliere dei giocatori.

TriClient

Per prima cosa il client controlla la correttezza degli argomenti forniti, in caso di argomenti non validi il client termina con un messaggio di errore che riporta il corretto uso del programma.

Il client controlla poi la presenza di un server in esecuzione, in caso non ci siano server attivi il client termina con un messaggio di errore avvisando l'utente che non ci sono server in esecuzione.

Il client procede poi a recuperare i semafori creati dal server e a collegarsi alla memoria condivisa. Prima di avvisare il server della sua entrata il client controlla che ci sia spazio nella partita per accoglierlo, in caso affermativo procede a comunicare al server la sua entrata, altrimenti procede a scollegarsi dalla memoria condivisa e a stampare un messaggio di errore che informa l'utente del fatto che non è stato possibile unirsi alla partita per poi terminare.

Infine se il client è ancora in esecuzione significa che è pronto a giocare, attende quindi un segnale di inizio partita da parte del server.

Approfondimento sulle funzionalità

Gestione della partita

La partita, come da consegna, è arbitrata dal server, il quale ha il compito di avvisare i client dell'inizio e della fine del loro turno, ad ogni fine turno controlla se la partita è finita, in caso affermativo avvisa i client di controllare il vincitore e di procedere a comunicarlo all'utente di conseguenza e termina, altrimenti fa iniziare un ulteriore round.

Esecuzione dei turni

I client finita l'inizializzazione si trovano in uno stato di attesa, quando il server percepisce entrambi i giocatori pronti procede a comunicare ai client che sta per partire un turno, quando i client sono entrambi pronti ad eseguire il turno sbloccano il server il quale procede ad avviarlo.
Ci sono 2 possibilità:

- **È il tuo turno:** il client comunica all'utente che è il suo turno. Stampa a video il simbolo del giocatore, la tabella di gioco attuale, una riga che riporta il modo in cui è possibile eseguire una mossa, un timer del tempo rimanente (solo visivo, indipendente dal timer del server che controlla i turni) e permette all'utente di fare la sua mossa (mosse non possibili e input non corretti vengono ignorati).
- **Non è il tuo turno:** il client comunica all'utente che non è il suo turno. Stampa a video il simbolo del giocatore, la tabella di gioco attuale e aspetta che l'altro giocatore faccia la sua mossa.

Una volta scaduto il tempo o fatta una mossa il server comunica ai client la fine del turno, questi ultimi procedono ad andare in uno stato di attesa, pronti a ricevere il via per il prossimo turno di gioco e così fino alla fine della partita.

Partita con il bot

Per avviare il bot, come richiesto da consegna, è necessario avviare il client specificando come argomenti sia il nome del giocatore che una flag (o "*"). Quando il client controlla gli argomenti, nota che è stato richiesto il bot, controlla quindi se ci sono 2 posti liberi nel server (altrimenti uno tra client e bot rimarrebbe fuori).

Una volta accertatosi che ci sia posto per loro, il client entra normalmente aggiornando però anche una flag "bot" presente in memoria condivisa atta a comunicare al server la richiesta di avvio del bot, il server quindi quando riceve il segnale di entrata del giocatore oltre ad aumentare il numero di giocatori controlla anche la flag "bot", in caso questa fosse a "true", si duplica con una fork() e procede ad eseguire il bot con una exec(). Il bot nonostante sia un eseguibile a parte è a tutti gli effetti un client e pertanto segue i protocolli di entrata e di esecuzione dei turni del client, infatti il server stesso non è in grado di riconoscere il bot da un normale client, l'unica differenza è appunto che il bot esegue in automatico mosse randomiche appena gli è possibile.

Scelte progettuali

- Abbiamo deciso di creare un eseguibile a parte per il bot per una semplice questione di pulizia del codice
- Se un giocatore non fa una mossa entro il tempo stabilito all'avvio del server il turno viene passato all'altro giocatore.
- A fine partita il server fa terminare i client (dopo aver comunicato loro i risultati e aver permesso loro di comunicarlo all'utente) e poi termina.
- in caso di abbandono da parte di un giocatore la partita finisce decretando il giocatore rimanente come vincitore e terminando la partita (disconnettendo anche il giocatore rimanente).
- Viene richiesto di chiedere doppia conferma per la chiusura del server con CTRL+C, noi abbiamo deciso di gestire il segnale eseguendo un thread che permette all'utente di terminare il programma premendo nuovamente CTRL+C entro 5 secondi dal primo comando, se entro questi 5 secondi l'utente non prosegue con il comando il server ripristina il funzionamento originale (ovvero si resetta la doppia conferma per la terminazione). All'utente (del server) viene comunicato il tempo rimanente per compiere queste azioni e anche il momento in cui il server ripristina il funzionamento originale. Questa operazione abbiamo deciso di eseguirla in un thread così da non essere bloccante per il resto dell'esecuzione del programma.
- I turni nel client vengono eseguiti in thread, in modo da renderli facili da terminare in caso di fine turno, disconnessioni, etc...
- Abbiamo deciso di abbellire un po' il programma e di sbizzarrirci un po' con le funzionalità degli ansi escape codes, questo ci ha permesso di rendere la nostra interfaccia utente meno noiosa e di muovere il cursore (anche se con un po' di difficoltà) nel terminale, così da poter avere dei timer in tempo reale dei vari turni e provare ad avere la tavola di gioco più stabile possibile in modo da rendere l'esperienza di gioco più gradevole.