Elaborato Sistemi Operativi

Davide Donà, VR485945
Andrea Blushi, VR485743

Giugno 2024

1 Funzionamento generale

1.1 TrisServer

L'eseguibile **TrisServer**, al suo avvio, esegue una serie di semplici operazioni:

1. Imposta il funzionamento dei **segnali** `SIGINT` e `SIGUSR1` (spiegati successivamente);
2. Controlla i parametri passati da terminale, segnalando all'utente l'uso corretto in caso di errori;
3. **Inizializza** l'area di **memoria condivisa** e i **semafori**, segnalando eventualmente la presenza di errori in tali operazioni.

Arrivati a questo punto, il **processo server** si pone in attesa dei processi client.

1.1.1 Memoria Condivisa

All'interno della memoria condivisa, abbiamo inserito una **struct** SharedData, contenente i campi necessari all'implementazione delle funzionalità. Essi sono:

- `char player[2]`: array contenente **2 char**. L'elemento in **posizione** i rappresenta il **simbolo** usato dal **giocatore** $i + 1$;
- `char playerName[2][STR_LEN]`: array contenente i **nomi** dei **client connessi**. La formattazione segue lo stesso schema del punto precedente;
- `int activePlayer`: indica il numero di client connessi. Viene aggiornato da ogni client;
- `char board[BOARD_SIZE]`: la **tabella di gioco**. Abbiamo deciso di utilizzare un **vettore** al posto di una **matrice** per questione di semplicità;
- `pid_t pids[NUM_PROCESSES]`: contiene i **PID** dei processi connessi. Necessario per lo **scambio di segnali**. La formattazione segue lo stesso schema visto in precedenza;
- `int stato`: rappresenta lo **stato** del gioco. Utilizzato in combinazione con i segnali in modo da permettere al server di comunicare diverse situazione, usando un unico segnale. I valori sono interpretati nel seguente modo:
  - **STATO == 0**: partita terminata in **PAREGGIO**;
  - **STATO == 1** o **STATO == 2**: partita **terminata**. **Vittoria** del giocatore con **indice == STATO**;
  - **STATO == 3**: disconnessione del server;
  - **STATO == 4**: disconnessione di uno dei due client;
- `int indexPlayerLefted`: indice del giocatore che ha abbandonato la partita;
- `int playAgainstBot`: indica se è stato scelto di giocare contro un bot;

1.1.2 Semafori

Per garantire la **MUTUA ESCLUSIONE** e la **SINCRONIZZAZIONE** tra i processi sono stati implementati **5 semafori**:

- **SEM_MUTEX**: utilizzato per garantire la **mutua esclusione** nell'accesso alla **memoria condivisa**;
- **SEM_PLAYER**: un semaforo per ogni giocatore. Permettono di garantire la sincronizzazione nei **round** tra i due processi client. Ad **ogni mossa eseguita**, sarà il **processo SERVER** a sbloccare il **client** successivo;
- **SEM_SERVER**: utilizzato per eliminare l'**attesa attiva** nel **processo server**. Questo permette al server di aspettare **passivamente** che il client attivo esegua la propria mossa, per poi essere risvegliato dal client stesso. Eseguirà a questo punto i controlli sullo stato della partita, prima di passare il turno al giocatore successivo;
- **SEM_INIZIALIZZAZIONE**: utilizzato durante la fase di inizializzazione, per permette al **processo SERVER** di risvegliarsi

1.2 TrisClient

L'eseguibile **TrisClient** può essere avviato solo successivamente all'avvio di un processo **TrisServer**. Infatti, è compito del TrisServer inizializzare la memoria condivisa. Nel caso in cui questa non fosse presente, il TrisClient rileva l'assenza e termina l'esecuzione. Sono eseguite una serie di operazioni di inizializzazione:

1. Imposta il funzionamento dei **segnali**.
2. Controlla i parametri passati da terminale, segnalando all'utente eventuali errori nell'uso.
3. Si **connette** all'area di **memoria condivisa** e ai **semafori** inizializzati dal TrisServer.

Per **garantire la mutua esclusione** nell'accesso alla memoria e **gestire** la **sincronizzazione**, si utilizzano i semafori descritti nel paragrafo precedente. Una volta collegato alla memoria condivisa, **TrisClient** verifica il numero di giocatori attivi prima di proseguire. Se viene superato il limite di giocatori, viene segnalato al nuovo processo e terminato direttamente.

Successivamente, se il giocatore ha scelto di giocare contro il bot, viene impostata ad 1 la variabile **playAgainstBot**. A questo punto il processo **TrisServer**, appena sbloccato, può creare un **processo figlio**, facendogli eseguire **TrisClient**. Tale processo, verificando la memoria condivisa, riconosce di essere un BOT e cambia di conseguenza il suo comportamento, generando le mosse automaticamente.

Infine, sono **inizializzati** i restanti valori della **memoria condivisa**: **playerIndex** (determinato dall'ordine di accesso), **pids** e **playerName**. Successivamente, il **TrisClient** sblocca il **TrisServer** e si pone di un riscontro.

Una volta terminata l'attesa, il processo entra nel ciclo principale. Esso gestisce l'alternarsi dei turni, stampando il tabellone aggiornato e richiedendo l' inserimento dell'input.

1.3 Segnali

Il normale flusso di esecuzione può essere **interrotto** da una **serie di segnali**, ciascuno con un comportamento specifico e una funzione importante nel contesto del gioco:

- **sigUser1Handler**: segnale generato dal **TrisServer**, assume comportamenti diversi a seconda della **variabile stato**, presente in memoria condivisa. In generale, gestisce tutti i casi che portano alla terminazione dei processi client. Il client, una volta ricevuto, stampa a video un messaggio che riassume lo stato del gioco, eseguendo infine una **terminazione sicura**.

  Tale segnale è inoltre generato dal processo TrisClient per comunicare al server una **disconnessione**. Sarà poi il server a segnalare al secondo TrisClient la sua vittoria, permettendo una **terminazione sicura** per tutti i processi;

- **sigAlarmHandler**: segnale generato dalla System Call `alarm`. Indica, al processo **TrisServer** che, il **giocatore attivo** ha terminato il proprio tempo per eseguire la mossa. Sarà compito del server passare il turno all'altro giocatore;

- **sigUser2Handler**: segnale generato dal processo **TrisServer**, rappresenta la scadenza del tempo assegnato per eseguire una mossa. Il client che lo riceve, si pone in attesa del prossimo turno.

- **firstSigIntHandler**: handler che interviene alla `SIG_INT` (pressione del tasto **Ctrl-C**). Bloccando la terminazione immediata del processo e visualizzato un messaggio che informa l'utente che una seconda pressione comporterà la terminazione. Sostituisce infine se stesso con il **secondSigIntHandler**

- **secondSigIntHandler**: gestisce il secondo segnale `SIG_INT`. A questo punto il processo esegue **terminazioneSicura**, ma solamente dopo aver comunicato la disconnessione agli altri processi o al server.