VERSIONE
PIPES + THREADS

SPACE DEFENDER

Sommario

Introduzione.

Cosa è un processo?

Un processo è un'entità dinamica caricata su memoria RAM generata da un programma. È identificato da un codice univoco chiamato PID e più precisamente, esso è una sequenza di attività (task) controllata da un programma (scheduler) che si svolge su un processore in genere sotto la gestione o supervisione del rispettivo sistema operativo.

Cosa è un thread?

Un thread è una suddivisione di un processo in due o più filoni (istanze) o sottoprocessi che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocessore (monothreading) o multiprocessore (multithreading) o multicore.

Il progetto di Sistemi Operativi.

Il progetto del corso, richiedeva la codifica di un programma chiamato "Space Defender" che è ispirato a due videogiochi arcade classici (Space Invaders e Defender). A fine sviluppo, Il gioco infatti presenta caratteristiche simili a questi due giochi. L'implementazione delle due versioni richieste dalle specifiche (processi + thread) consentono di mettere in pratica ciò che è stato visto a lezione, affrontando i principali problemi della programmazione concorrente di sistema, ovvero la sincronizzazione.

Figura 1-Errore di sincronizzazione

I processi comunicano tra di loro tramite le pipes di linux, dei file descriptor a cui ci si può accedere per leggere e scrivere come se fossero una sorta di file. Al loro interno distinguiamo 3 canali principali: lettura/scrittura/errori.

I thread invece possono comunicare tra di loro mediante una zona di memoria condivisa, ovvero un buffer avente una determinata grandezza in cui vengono messi/prelevati i dati.

Le regole del gioco.

- *Se il giocatore perde tutte le vite a sua disposizione poiché viene colpito ripetutamente dalle bombe, oppure gli alieni arrivano a toccare il giocatore invadendolo, si perde.*
- *Se il giocatore riesce a sopravvivere evitando le bombe che sparano i nemici, e riesce tramite i suoi proiettili a distruggere tutte le navicelle nemiche, si vince.*

La libreria ncurses.

Per lo sviluppo del progetto, è stata utilizzata la libreria grafica ncurses, la quale copia la memoria del terminale di linux modificandola per mostrare graficamente tramite le sue funzioni, ciò che si vuole stampare.

Suddivisione dei file nel progetto.

Il nostro progetto è stato diviso nei seguenti file (assieme ai relativi .h che contengono le definizioni):

- **libProgetto.h**: contiene la stuttura dati di tipo pos, utilizzata dalle entità che si muovono nel programma, le misure dello schermo e anche il parametro M contraddistinto da una define M che consente di scegliere il numero dei nemici da generare.
- **makefile**: contiene le direttive che consentono, tramite il comando make, di compilare il programma creando l'eseguibile spdef che può essere lanciato dal terminale una volta compilato, con ./spdef.
- **main.c**: questo file contiene le istruzioni che permettono di inizializzare lo schermo per utilizzare la libreria grafica ncurses. Oltre a fare ciò, viene richiamato qua anche il menu che consente di richiamare tutte le funzioni all'interno del programma.
- **start.c**: qua dentro troviamo la creazione delle pipe che consente la comunicazione interprocesso e la creazione dei processi base che permettono al gioco di funzionare (1 processo di controllo, 1 processo giocatore, M processi nemici).
  Nella versione thread, si creano invece i thread e si inizializza il mutex che consente di avere una corretta sincronizzazione tra i processi.
- **menu.c**: contiene il codice per le animazioni iniziali del gioco e la stampa del menu.
- **oggetti.c**: contiene le funzioni di ciascun oggetto che si vede muoversi sullo schermo.

Per lo sviluppo si è scelto di seguire una filosofia di semplicità implementativa e funzionale. Inoltre si è sfruttato l'approccio del divide-et-impera. Tale approccio si evince molto nelle funzioni che utilizzano delle sotto funzioni per implementare una determinata funzionalità del programma (es. opening che sfrutta ufo e welcome).

Come avviare il progetto e requisiti di sistema.

Il progetto è stato sviluppato sul sistema operativo Ubuntu Linux 64 bit, e per questo a lui è destinato. Potrebbe essere necessario installare ncurses nel sistema Ubuntu sul quale si sta compilando tale progetto. ***ATTENZIONE: si consiglia di impostare il terminale ad una grandezza pari a 100x30 per evitare bug grafici durante l'esecuzione del programma***. Per effettuare le seguenti operazioni è necessario accedere al terminale del sistema operativo ed eseguire i seguenti comandi:

Figura 2 - come impostare il terminale

Installare ncurses:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Installare la libreria build-essential, che consente di avere tutti i riferimenti necessari per costruire un pacchetto. Contiene il compilatore, ma anche il debugger.

```
sudo apt-get install build-essential
```

Avviare il progetto:

aprire il terminale nella cartella contenente i file del progetto che si vuole avviare e lanciare il comando make, il quale legge le istruzioni del makefile presente nel progetto per costruire l'eseguibile del programma "spdef".

```
make
```

Pulire tutti i file oggetto generati, lasciando solo il file eseguibile:

```
make clean
```

Avviare il file eseguibile appena buildato:

```
./spdef
```

VERSIONE PROCESSI.

Struttura della versione dei processi.

Il programma è strutturato seguendo una struttura di 1 consumatore e N produttori. I produttori sono le varie entità che in ciascuna funzione producono delle coordinate che vengono inviate alla pipe principale; il processo consumatore invece, legge dalla pipe disegnando sullo schermo del terminale i vari oggetti. Il processo consumatore, inoltre, essendo un processo che controlla l'andamento del gioco si occupa di controllare infatti le collisioni tra gli oggetti, in maniera tale da applicare le regole del gioco, vincendo o perdendo di conseguenza.

---

main.c

- ***int main()***: è il main del nostro programma. Si occupa di inizializzare lo schermo per le ncurses e di richiamare il menu che consente di richiamare tutte le altre funzioni del programma.

menu.c

- ***void generaStelle(WINDOW *finestra)***: genera le stelle sfruttando la funzione rand() all'interno di una finestra. Tale finestra, deve essere passata in input come parametro, dopo che è stata creata con l'apposita funzione di ncurses che rende appunto un puntatore di tipo WINDOW associato alla finestra stessa.
- ***void stampaMenu()***: stampa il menu del gioco, il quale prende in input un intero. Tale intero deve appartenere a quelli mostrati nel menu del gioco per chiamare le funzioni corrispondenti alla scelta fatta dall'utente.

Figura 3 - menu del gioco

- ***void crediti()***: mostra gli autori del gioco e lo scopo per il quale è stato creato.
- ***void comandi()***: tale funzione spiega all'utente i comandi che deve usare per giocare a Space Defender.
- ***void story()***: questa funzione stampa a schermo una sorta di mini storia introduttiva al gioco, sfruttando come funzione di supporto, generaStelle(WINDOW *finestra).
- ***void combatti()***: altra funzione che aggiunge il secondo pezzo di storia introduttiva al gioco, invitando l'utente all'azione.

- ***void attenzione()***: messaggio informativo di inizio gioco, che informa di impostare la finestra del terminale a 100x30 per evitare bug grafici.
- ***void opening()***: questa funzione sfrutta le funzioni welcome() e ufo() per costruire l'animazione iniziale del gioco.
- ***void welcome()***: stampa il messaggio "welcome" gradualmente.
- ***void ufo()***: fa muovere un ufo nello schermo che lancia un raggio e poi se ne va.

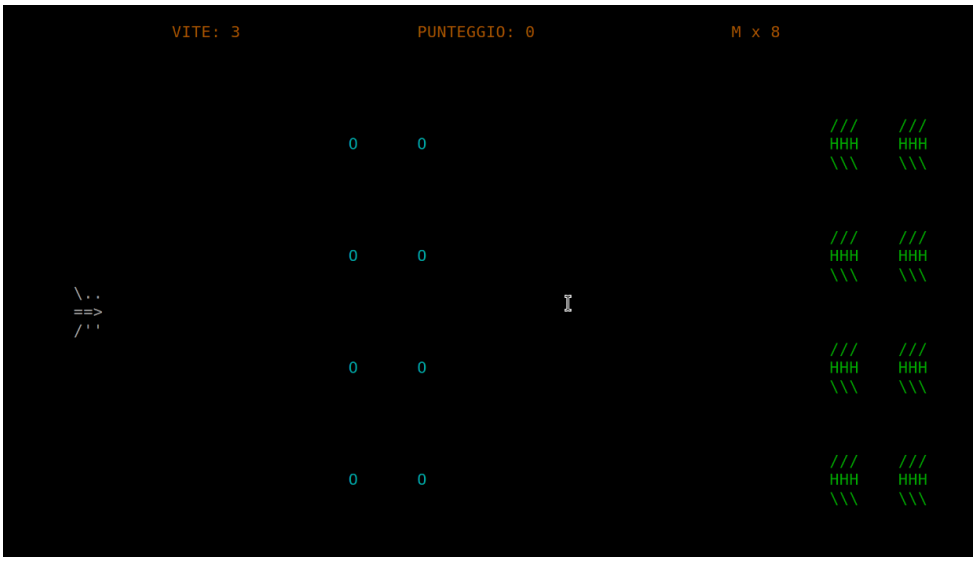

Figura 4 - il gioco in esecuzione

oggetti.c

- ***void stampaOggetto(WINDOW *finestra, int dim_sprite, char oggetto[][dim_sprite], int posx, int posy, int mode)***: consente di stampare/canellare gli oggetti nello schermo prese le loro coordinate e la finestra in cui stampare.
- ***void giocatore(int pipeOutCoordinate, int pipeOutNavicella)***: funzione che legge l'input da tastiera muovendo la navicella o dando l'input per stampare i due missili.
- ***void missile(pos pos_giocatore, int pipeOutCoordMissile)***: genera le coordinate dei due missili che spara la navicella del giocatore.
- ***void nemico(int pipeOutCoordinate, int pipeInNemici)***: genera le coordinate di una navicella nemica.
- ***void bomba(pos nemico, int pipeOutCoordinate)***: genera le coordinate di una bomba che parte da un nemico.
- ***void controllo(int pipeInCoordinate, int pipeOutNemici, int pipeOutNavicella, int pipeOutCollisioni)***: controlla tutte le coordinate ricevute dalle altre funzioni stampando gli oggetti, gestendo la collisione degli oggetti e regola il funzionamento del gioco in generale.
- ***void refreshScoreFile(int score)***: crea/legge il file score.dat che contiene l'informazione dei punteggi ottenuti dal giocatore durante la partita.
- ***void gameover()***: stampa la schermata di game over.
- ***void gamewon()***: stampa la schermata di vincita del gioco.
- ***void youlose()***: feedback vocale perdita gioco.
- ***void youwon()***: feedback vocale vincita gioco.

- ***_Bool esitoGioco(int vite, int punteggio, int nemiciRimanenti, pos *nemici)***: funzione che restitusice un booleano (true se il gioco può continuare, falso altrimenti). Tale funzione si basa sulla funzione isInvaso().
- ***_Bool isInvaso(pos *nemici)***: dice se le navicelle nemiche hanno toccato il giocatore o meno restituendo true o false. La restituzione di true causerebbe la fine del gioco nel caso avvenisse appunto l'invasione da parte dei nemici.
- ***void muoviNemici(pos nemico, int pipeOutCoordinate)***: funzione che prende in input un nemico e poi lo anima generando le sue coordinate.
- ***void inizializzaNemici(pos *nemici, int nemicoX)***: inizializza i campi del vettore dei nemici.
- ***void inizializzaBombe(pos *bombe)***: inizializza i campi del vettore di bombe.
- ***void inizializzaMissili(pos *missili)***: inizializza i campi del vettore di missili.
- ***int generaNumero(int min, int max)***: genera un numero compreso tra min e max presi come parametri.
- ***void soundsEffectManager(int mode)***: riproduce i suoni degli eventi di gioco e prende come parametro la macro che indica il suono da riprodurre.

start.c

- ***void start()***: crea i processi principali del gioco.

VERSIONE THREADS.

Struttura della versione threads.

Il programma è strutturato seguendo una struttura di 1 consumatore e N produttori. I produttori sono le varie entità che in ciascuna funzione producono delle coordinate per modificare le variabili globali; il thread consumatore invece, disegna sullo schermo del terminale i vari oggetti. Il thread consumatore, inoltre, essendo un thread che controlla l'andamento del gioco si occupa di controllare infatti le collisioni tra gli oggetti, in maniera tale da applicare le regole del gioco, vincendo o perdendo di conseguenza.

La versione dei thread, differisce da quella sui processi, dall'introduzione di un mutex che consente di proteggere le aree critiche di codice. Tali sezioni critiche, che sono le zone in cui il codice scrive sulla memoria in concomitanza con altri thread e se non vengono modificate da un thread alla volta, possono produrre errori nei dati contenuti dentro alle variabili. È per questo che usiamo i mutex: ci aiutano infatti a capire se un thread sta scrivendo dei dati e quindi il thread concorrente aspetta che l'altro thread termini l'accesso per accedervi lui stesso.

Modifiche alle specifiche introdotte.

- I nemici cambiano colore se colpiti. Ogni colore indica le vite rimanenti (verde = 3, giallo = 2, rosso = 1).
- Inserito il conteggio dei punteggi per ogni hit, uccisione di un nemico, penalità se colpiti da una bomba nemica.
- Inserita la soundtrack e gli effetti sonori relativi agli eventi di gioco.
- Inserita una storia introduttiva con varie schermate che presentano il gioco.
- Grafica migliorata con piccole animazioni.

SOUNDTRACK.

La soundtrack è stata remixata/tagliata/renderizzata tramite il software libero open source Audacity.