Universitità di Cagliari

Facoltà di Scienze MM.FF.NN.

Tesina Corso SO1 - Parte 3, SystemV

Davide Gessa (45712)

A.A. 2011 - 2012

7 Gennaio 2012

Indice

Capitolo 1

Struttura del programma

1.1 Suddivisione files

Il codice del programma è stato ripartito in vari file cercando di dividere le varie funzionalità dei diversi oggetti della scena:

- alien.c alien.h - navicella aliena
- bomb.c bomb.h - bomba aliena
- control.c control.h - controllo delle iterazioni tra i vari oggetti e rendering della scena
- main.c - starter dell'applicazione
- missile.c missile.h - missile lanciato dall'astronave
- scores.c scores.h - gestione dei punteggi
- space_ship.c space_ship.h - navicella giocatore
- utility.c utility.h - funzioni varie utili per il funzionamento del programma
- spaceinvaders.h - definizioni globali

1.2 Elenco funzioni

- alien_task() : gestione navicella aliena
- bomb_task() : gestione bomba aliena
- clear_quad() : cancella un area quadrata dello schermo
- control_check_collision() : controlla se c'è una collisione tra due oggetti

- control_task() : gestione della scena
- missile_task() : gestione di un missile
- render_string_array() : renderizza uno sprite nello schermo
- scores_add() : aggiungi un punteggio alla lista punteggi
- scores_load() : carica i punteggi da un file
- scores_save() : salva i punteggi in un file
- space_ship_task() : gestione navicella giocatore
- timevaldiff() : calcola la differenza tra due strutture timeval

1.3 Il main

La funzione main si preoccupa di fare le fork per gli oggetti iniziali, ed avviare le relative funzioni di gestione.

1. Creazione processo navicella
2. Creazione dei processi degli alieni
3. Avvio della funzione di controllo

Capitolo 2

Architettura del programma

Ogni oggetto segue un funzionamento simile agli altri oggetti, e viene creato nel medesimo modo:

1. Il padre fa una fork e continua la sua normale esecuzione
2. Il figlio avvia la funzione di gestione dell'oggetto in questione; le funzioni di gestione hanno come nome oggetto_task(...) e come argomenti le informazioni relative al posizionamento.

La funzione di gestione di un oggetto è così strutturata:

1. Inizializza i dati iniziali e li inserisce in una struttura di tipo object_data_t
2. Ricava la chiave e la utilizza per ricavare la coda dei messaggi globale, per l'invio della posizione
3. Se è un alieno, ricava la chiave relativa ad esso e la utilizza per ricavare la coda dei messaggi delle collisioni
4. Esegue un loop (dal quale esce al verificarsi di alcune condizioni) che ad ogni iterazione produce le nuove informazioni dell'oggetto (ad esempio, tramite la tastiera nel caso della navicella del giocatore), le inserisce nella sua struttura e le invia al controllo tramite le api SystemV. Inoltre, il controllo può inviare informazioni ai vari oggetti per segnalare un avvenuta collisione (vedi sezione 2.1).

Le informazioni di un oggetto sono memorizzate nella seguente struttura:

```
///> Struttura contenente le informazioni relative all'oggetto
typedef struct
{
long  mtype; ///< Tipo di messaggio, usato dalle system V
int  x; ///< Posizione x dell'oggetto
```

```
int y; ///< Posizione y dell'oggetto
int  size; ///< Dimensione dell'oggetto (sia x che y)

object_type_t type; ///< Tipo di oggetto
int life; ///< Vita rimanente all'oggetto

pid_t pid; ///< Pid dell'oggetto
int id; ///< Id locale dell'oggetto (usato per creare la queue dei messaggi)
} object_data_t;
```

2.1 Comunicazione dei Task

Per quanto riguarda la comunicazione tra il task di controllo e gli oggetti, ho strutturato il software in modo che ci fosse il minor numero di code di messaggi, per diminuire l'occupazione di memoria, ed il numero di msgrcv/msgsnd; tutti gli oggetti della scena condividono la stessa coda di messaggi che utilizzano per inviare i propri dati al controllo (tutti gli oggetti ci scrivono, e solo il controllo ci legge).

Inoltre, ogni alieno ha un altra coda di messaggi, condivisa con il controllo; viene utilizzata dal controllo per inviare lo stato delle collisioni dell'alieno; il numero di code di messaggi utilizzate nel software è quindi pari a (NUMERO_ALIENI + 1).

Per gli altri oggetti, avendo la necessità di segnalare un avvenuta collisione, ho utilizzato il meccanismo dei segnali come ho fatto nel progetto delle pipes. Se un oggetto deve essere distrutto perchè è avvenuta una collisione, il controllo gli invia un segnale SIGNQUIT; ogni oggetto ha al suo interno una funzione che gestisce questo segnale, e rende falsa la condizione del loop.

Una schematizzazione della comunicazione tra i task dell'applicazione è visibile in figura 2.1.

2.2 Tasks

Come stabilito nelle specifiche, alieni, controllo, bombe, navicella e missili utilizzano un processo separato ciascuno.

2.2.1 Controllo

Il processo di controllo esegue un loop, ed ad ogni iterazione si occupa di:

1. Ricevere dalla coda di messaggi condivisa le informazioni relative ad un oggetto

2. Salvare in un array le informazioni ricevute

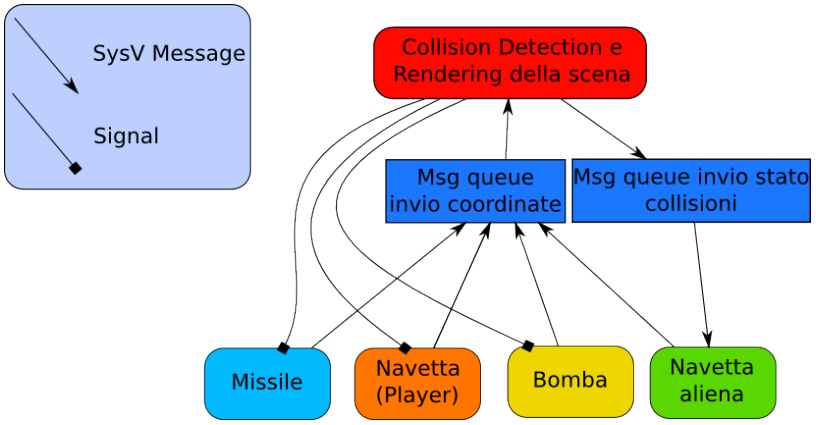

Figura 2.1: Schema di comunicazione fra i tasks

3. Controllare se l'oggetto è in collisione con un altro oggetto, ed in tal caso, eseguire un azione appropriata

4. Cancellare l'area di schermo occupata dall'oggetto nell'iterazione precedente

5. Ridisegnare l'oggetto nella nuova posizione

Il ciclo viene interrotto quando si verifica una condizione di gameover, o quando il giocatore vince. Il processo di controllo è il padre di tutti gli oggetti della scena; quando il gioco finisce, il controllo invia a tutti i processi ancora in vita (missili, bombe, alieni, navicella) un messaggio (che può essere un opportuno messaggio nella coda dei messaggi delle collisioni, per quanto riguarda gli alieni, o un segnale SIGQUIT per tutti gli altri oggetti) per avvisare che il programma deve chiudersi.

Finito il gioco, viene aggiunto il punteggio nella lista punteggi salvata in un file, e viene visualizzata la classifica.

2.2.2 Alieno

Si muove seguendo un percorso destra->giu->sinistra->giu come nel gioco originale; ad ogni iterazione invia al controllo la nuova posizione, e riceve tramite una coda di messaggi non bloccante, le informazioni relative alle collisioni: se collide con un altro alieno, cambia direzione, altrimenti decrementa la vita, e nel caso abbia finito le vite disponibili, si distrugge e si ricrea di livello superiore.

Ad intervalli regolari l'alieno sgancia una bomba, generando un figlio.

2.2.3 Bomba

Il processo bomba ad ogni iterazione, scende di una posizione, invia le info al controllo, e attende un tempo predefinito; il loop termina quando riceve un segnale SIGNQUIT dal processo di controllo.

2.2.4 Navicella

Ad ogni iterazione, il processo della navicella attende un input da tastiera per comandare il movimento o sparare dei missili; il movimento può essere solo orizzontale. Il lancio dei missili è limitato da un timer, che permette un certo numero di spari ogni secondo. Premuto il tasto di sparo, la navicella crea due nuovi processi (missile destro e missile sinistro) che avviano la funzione di gestione del missile.

Missile

Il processo missile ad ogni iterazione, sale di una posizione in verticale, si sposta di un'unità in orizzontale (a seconda che sia un missile destro od un missile sinistro), invia le info al controllo, e attende un tempo predefinito; come per la bomba, il loop termina quando riceve un segnale SIGNQUIT dal processo di controllo.

2.3 Sincronizzazione

Non sono state utilizzate primitive di sincronizzazione, in quanto i processi non hanno dati condivisi. La comunicazione avviene tramite code di messaggi (systemV) e segnali (posix), il kernel linux si occupa della sincronia e dei dati condivisi.