Università di Cagliari
Facoltà di Scienze MM.FF.NN
Corso di Laurea in Informatica

TESINA CORSO SISTEMI OPERATIVI 1

Docente S. Carta (salvatore@unica.it)
Tutor L. Boratto (ludovico.boratto@unica.it)
A.A. 2011 -2012

GRUPPO

MARIO ROSSI NM xxxxx
GIUSEPPE BIANCHI NM yyyyy

specifica versione 1.1

1 Modalità di consegna e di esame

- La tesina si compone di 3 esercizi, da svolgere secondo le specifiche definite al punto 2. I primi due esercizi sono obbligatori e daranno diritto a 4 punti del voto finale, il terzo esercizio è opzionale (completando la tesina con tutti e 3 gli esercizi sarà possibile ottenere fino a 6 punti). Si veda il file "Modalità d'esame" presente nel Materiale Didattico per ulteriori dettagli sul calcolo della media.
- Il terzo esercizio può essere presentato fino all'appello di febbraio 2012 (negli appelli successivi sarà possibile consegnare e discutere solo i primi due esercizi del progetto).
- Tutto il materiale in formato elettronico (questo file non zippato, più 3 cartelle contenti i sorgenti denominate esercizio1, esercizio2 ed esercizio 3, sotto forma di archivio zippato), dovrà essere spedito all'indirizzo e-mail lso.unica@gmail.com, entro la data di consegna stabilita per ciascun appello, indicata nel sito informatica.unica.it, sezione "Prenotazione esami".
- L'email dovrà avere il seguente titolo: "[SO1] Tesina 2011-2012 - Mario Rossi – Giuseppe Bianchi"
- Il codice sorgente deve essere presentato all'esame.
- Il codice sorgente deve essere fornito su supporto cd-rom, corredato di opportuni makefile. Al momento dell'esame ciascuno studente avrà a disposizione un PC con Linux (Ubuntu), e sarà sua cura inserire il cd, e procedere alla compilazione ed esecuzione del codice. **Si potrà utilizzare solo comandi da shell Bash, non si potranno utilizzare comandi in modalità grafica** (fa parte dell'esame).
- La parte di esame riferita alla tesina consisterà nell'accertamento:
    - del corretto funzionamento del codice, della qualità dei commenti e della documentazione
    - della capacità dello studente di giustificare una qualunque parte del codice
    - della capacità dello studente di manipolare il codice in modo da modificarne opportunamente la funzionalità
    - Della capacità dello studente di giustificare qualunque scelta progettuale compiuta ed eventualmente di argomentare eventuale variazioni proposte dal docente (p.e. "perchè due pipe anziché una?" oppure "se avessimo usato una lista linkata al posto di un buffer di memoria come sarebbe eventualmente cambiata la gestione della sincronizzazione tramite semafori e mutex", ecc.)
- Ciascun componente di un gruppo sarà interrogato separatamente dagli altri.

2 Programmazione di sistema in Linux: Implementazione di una funzionalità grafica utilizzando strumenti per la programmazione parallela in ambiente Linux in almeno 2 diverse versioni

2.1 TESTO

Codificare, compilare ed eseguire su Linux un programma che implementi una funzionalità grafica ispirata ad un antico videogame (nome originale ***Space Invaders***). Utilizzare gli strumenti per la elaborazione parallela, la comunicazione e sincronizzazione visti a lezione. Riferirsi all'**esercizio guardie e ladri** per avere un'idea di una possibile infrastruttura di base del programma.

- Visualizzare sullo schermo un oggetto astronave, il cui movimento è limitato a spostamenti laterali sulla linea più in basso dello schermo. L'astronave è rappresentata da una forma a scelta che occupi un'area massima di 6x6 caratteri. Il suo movimento è definito dai tasti direzionali della tastiera. Utilizzare un thread o processo (a seconda della versione) che implementi la generazione delle coordinate dell'oggetto astronave.

- L'astronave, ogni qualvolta viene premuto il tasto spazio, rilascia 1 oggetto missile rappresentato da un singolo carattere che si spostano dal basso verso l'alto in maniera diagonale, uno verso destra e uno verso sinistra. Utilizzare thread o processi (a seconda della versione) che implementino la generazione delle coordinate degli oggetti missile.

- Visualizzare sullo schermo m (m è un parametro dell'applicativo) oggetti (navicelle nemiche) che si muovono dall'alto verso il basso e da sinistra a destra o da destra a sinistra a destra, non necessariamente in formazione allineata. Ciascun oggetto occupi un'area di 3x3 caratteri. E' richiesto venga utilizzato un thread o processo distinto (a seconda della versione) che implementi la generazione delle coordinate di ciascun singolo oggetto astronave, fino a che ciascuna astronave viene completamente distrutta. Una navicella singola è distrutta quando viene colpita da 2 missili lanciati dalla astronave. Questo tipo di gestione rappresenta la prima fase della vita delle navicelle nemiche. Quando la traiettoria di navicella incrocia quella di un'altra navicella, le traiettorie invertono il moto in senso orizzontale, provocando un effetto rimbalzo.

- La prima fase di vita di una navicella nemica finisce quando viene distrutta. A questo punto per ogni navicella distrutta si crea una nuova navicella nemica (navicella di secondo livello) che occupi un'area di 4x4 caratteri, avente movimento a piacere. Utilizzare un nuovo thread o processo (a seconda della versione) per la generazione delle coordinate di ciascuna nuova navicella nemica. Quando la traiettoria di navicella incrocia quella di un altra navicella, le traiettorie invertono il moto in senso orizzontale, provocando un effetto rimbalzo. Una navicella di secondo livello è distrutta quando viene colpita da 3 missili lanciati dalla astronave.

- La seconda fase di vita di una navicella nemica finisce quando viene distrutta. A questo punto per ogni navicella distrutta si crea una nuova navicella nemica (navicella di terzo livello) che occupi un'area di 5x5 caratteri, avente movimento a piacere. Utilizzare un nuovo thread o processo (a seconda della versione) per la generazione delle coordinate di ciascuna nuova navicella nemica. Quando la traiettoria di navicella incrocia quella di un altra navicella le traiettorie invertono il moto in senso orizzontale, provocando un effetto rimbalzo. Una navicella di terzo livello è distrutta quando viene colpita da 4 missili lanciati dalla astronave.

- Ciascuna navicella nemica (in tutte le fasi di vita) rilascia ad intervalli regolari un oggetto (bomba) rappresentato da un singolo carattere che si sposta dall'alto verso il basso in maniera rettilinea. Utilizzare un thread o processo (a seconda della versione) che implementi la generazione delle coordinate dell'oggetto bomba.

- Se l'astronave viene colpita da una bomba (o anche da più bombe, a piacere) o entra in contatto con una navicella nemica esplode.

- Il gioco finisce se:
  - Tutte le navicelle nemiche sono distrutte (il giocatore vince)
  - L'astronave viene distrutta (il giocatore perde)
  - Una navicella nemica raggiunge la linea più in basso dello schermo (il giocatore perde)

- E' obbligatorio che l'architettura del programma sia basata su un task (thread o processo a seconda della versione) per ciascun oggetto che si muove sullo schermo (astronave, navicella nemica di primo, secondo o terzo livello, missile, bomba). Ciascuno di questi task si deve occupare di generare e trasmettere le proprie coordinate ad un task che si occupa di: (1) disegnare sullo schermo gli oggetti; (2) verificare le collisioni o l'uscita di un oggetto dallo schermo e comportarsi di conseguenza. Il cuore del programma è incentrato sulla comunicazione delle coordinate fra *n* produttori (gli oggetti in movimento) e un consumatore (gestore di disegno e movimento). Verificare sempre che siano evitate le scritture su buffer di comunicazione pieno e le letture da buffer di comunicazione vuoto, per ciascuno degli esercizi.

- Una architettura simile che si può usare come base di partenza e dalla quale si può copiare la gestione della grafica è quella dall'esercizio guardie e ladri. In quell'esercizio ciascun oggetto ha un task associato che genera la sua posizione in maniera random, o secondo una traiettoria, o seguendo i tasti. Vi sono poi task che si occupano di disegnare sullo schermo gli oggetti che si spostano (prima si cancella la vecchia posizione, poi si disegna la nuova). Vengono inoltre monitorate le collisioni.

- E' opportuno scegliere correttamente la durata delle pause negli spostamenti degli oggetti, in modo da avere una visualizzazione comprensibile. Tenere conto della specifica particolare richiesta per la prima fase di vita delle navicelle nemiche.

- Opzionalmente saranno estremamente apprezzate versioni raffinate delle specifiche, sulla falsariga del videogame originale o di altri simili:
  - astronave e navicelle che non esplodono al primo colpo, ma perdono un carattere ogni volta
  - rocce sopra l'astronave che la proteggono dalle bombe e si disgregano mano a mano vengono colpite
  - bombe o missili particolari che esplodono facendo più danni
  - definizione di un punteggio per il giocatore
  - quadri successivi del gioco
  - posizionamento del gioco con navicelle a sinistra e astronave a destra che si può spostare anche in avanti
  - astronave che incrementa gli armamenti e le protezioni mano a mano che si va avanti
  - chi più ne ha più ne metta…

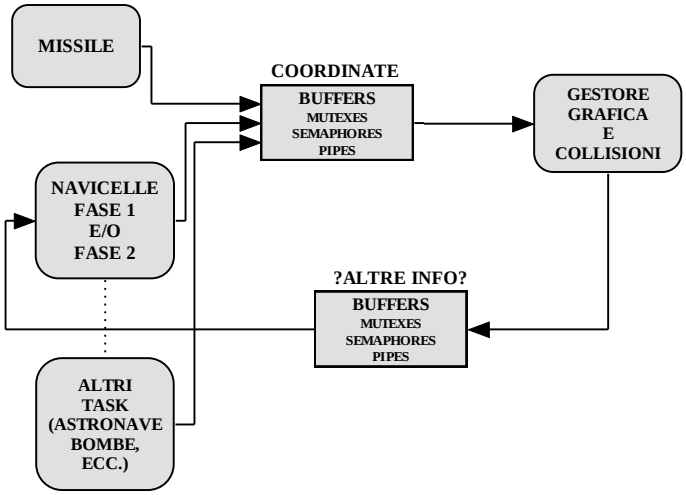
Versione 1 (obbligatoria): Utilizzare i processi per la gestione della elaborazione concorrente e le pipes per la comunicazione (simile all'esercizio guardie e ladri)
Versione 2 (obbligatoria): Utilizzare i thread per la gestione della elaborazione concorrente, un buffer in memoria condivisa per la comunicazione e gli strumenti di sincronizzazione delle libreria pthread (usare come riferimento gli esercizi produttore-consumatore visti a lezione)
Versione 3 (opzionale): Utilizzare i processi per la gestione della elaborazione concorrente, buffer in memoria condivisa (usare come riferimento gli esercizi produttore-consumatore visti a lezione) e gli strumenti di sincronizzazione e di comunicazione della System V (usare come riferimento il Gapil).

2.2 Documentazione Versione 1 (max 2 pagine in tutto)

1. Descrizione struttura del programma (elenco delle funzioni e descrizione sommaria del main)

2. Descrizione architettura del programma

- Descrizione della finalità e della implementazione di ciascun task
- Descrizione delle primitive di sincronizzazione utilizzate (che problemi risolvono e perché)
- Descrizione grafica della comunicazione fra i task.



ESEMPIO DESCRIZIONE COMUNICAZIONE

2.3 Documentazione Versione 2 (max 2 pagine in tutto)

1. Descrizione struttura del programma (elenco delle funzioni e descrizione sommaria del main)

2. Descrizione architettura del programma

- Descrizione della finalità e della implementazione di ciascun task
- Descrizione delle primitive di sincronizzazione utilizzate (che problemi risolvono e perché)
- Descrizione grafica della comunicazione fra i task.

2.4 Documentazione Versione 3 (max 2 pagine in tutto)

1. Descrizione struttura del programma (elenco delle funzioni e descrizione sommaria del main)

2. Descrizione architettura del programma

- Descrizione della finalità e della implementazione di ciascun task
- Descrizione delle primitive di sincronizzazione utilizzate (che problemi risolvono e perché)
- Descrizione grafica della comunicazione fra i task.