CORSO SO (B – T3) 2017/2018

Progetto Unix: simulazione di una società di individui

Università di Torino - Dipartimento di Informatica

Componenti gruppo : Emilian Patrasc

Matricola : 797859

1 Progetto Unix

1.1 Introduzione

Il presente documento ha lo scopo di presentare brevemente il programma sviluppato e le scelte progettuali effettuate.
Il progetto è stato interamente realizzato da Emilian Patrasc.

L'applicativo e composto da 3 eseguibili e un file "libreria" che condivide delle funzionalità comuni ai tre.

I 3 eseguibili sono ottenuti dalla compilazione dei seguenti sorgenti:

- gestore.c: processo gestore
- individuo_a.c: processo di tipo A
- individuo_b.c: processo di tipo B

Questi condividono strutture e funzioni che sono presenti nel file library.h.

1.2 gestore.c

Il gestore inizia leggendo i parametri input, questi possono essere imputati da linea di commando oppure letti da apposito file di configurazione (./config/config.csv). Se viene passato almeno un parametro da linea di commando (escludendo il primo parametro default che è il path dell'eseguibile) viene ignorato il file di configurazione.

Settati i 4 parametri input: *INIT_PEOPLE*, *GENES*, *BIRTH_DEATH* e *SIM_TIME*, inizia l'esecuzione del programma.

Vengono generati *INIT_PEOPLE* tramite *fork* e messi in attesa tramite *pause()*.

Finita la generazione di tutti i processi il gestore rilascia un segnale *SIGUSR1* gestito dalla funzione *wake_up*_process che innesca lo sblocco di tutti i processi figli; tramite una *execv* ed in base al tipo eseguono il file compilato *individuo_a.exe* o *individuo_b.exe.*

Le informazioni dei processi tipo *A* vengono salvati e condivise su memoria condivisa *IPC*, gli accessi vengono gestiti da un apposito semaforo creato dal gestore.

Per gestire gli eventi *BIRTH_DEATH e* SIM_TIME, il gestore utilizza la funzione *alarm()* e maschera il segnale *SIGALRM* tramite *sigaction* in modo da poter gestire concatenamenti di segnali.

La funzione *alarm_handler* gestisce il segnale SIGALRM e tramite due contatori tiene traccia dei due eventi. Ogni BIRTH_DEATH viene terminato un processo scegliendone uno casuale e a SIM_TIME secondi termina la simulazione. Questo avviene settando a 1 la variabile *end_simulation* di tipo "volatile", questo permette di rendere l'operazione atomica, quindi eseguita in un'unica istruzione.

Il gestore invia un segnale SIGTERM a tutti i processi figli per segnalare la fine della simulazione e gestore tramite la funzione *wait()* tiene traccia della terminazione di tutti i processi, in modo da evitare di avere processi zombie. Inoltre, utilizzando una coda di messaggi, legge i pid comunicati dai processi figli che hanno trovato un match con un processo di tipo opposto, gestendo così la creazione di nuovi processi.

Prima di terminare, il processo gestore, libera tutti gli allocamenti di memoria ed elimina la memoria condivisa, il semaforo e la coda di messaggi.

1.3 individuo_a.c

Il processo inizia leggendo i parametri *argv* nome e genoma passati dalla funzione *execv*. Crea due maschere per i segnali *SIGTERM* e *SIGALRM*. Il primo serve per gestire la terminazione del processo in modo da non avere un stato inconsistente, il secondo per gestire un timeout sulle risorse fifo, cosi da evitare processi bloccati in open dei file.

Il processo entra in un loop e apre in lettura un file fifo con nome il pid, mettendosi così in attesa di essere contattato da un processo di tipo B. Quando questo avviene, legge le informazioni inviate e decide se il processo ha un genoma abbastanza elevato, sia con risposta positiva o negativa risponde al processo mittente aprendo una fifo in scrittura utilizzando come nome il pid di quest'ultimo.

Se la risposta è negativa si rimette in attesa di essere contattato da un altro processo con lo stesso meccanismo. Con risposta positiva il processo contatta il gestore inviando, tramite coda di messaggi, il pid A ed il pid B.

Viene liberata la memoria e il processo termina.

In caso il gestore invii il segnale SIGTERM, l'esecuzione non termina immediatamente ma la variabile *done*, che controlla il loop principale, viene setttata a 1 e vengono eseguite tutte le restanti istruzioni fino alla fine, questo permette di avere una chiusura con uno stato consistente.

1.4 individuo_b.c

Il processo inizia con la stessa logica del processo di tipo A, lettura dei parametri input e creazione delle due maschere per i segnali SIGTERM e SIGALRM.

Dopodiché accede alla memoria condivisa per la lettura delle informazioni dei processi di tipo A e ricerca il processo adatto per essere contattato, trovato, contatta il relativo processo con il meccanismo delle fifo.

Per gestire la concorrenza viene creato un semaforo per ogni file fifo di tipo A e solo un processo per volta può accedere per la scrittura.

Se il processo A risponde affermativo contatta il gestore inviando un messaggio tramite coda di messaggi con pid di B e pid di A, esce dal loop ed esegue le operazioni di pulizia finali.

La gestione del segnale SIGTERM avviene con lo stesso meccanismo del processo A.