Progetto sistemi operativi 2023/2024

Partecipanti:
Raffaele Melluso 1045495 raffaele.melluso@edu.unito.it
Taoufik Ouhadid 1048137 taoufik.ouhadid@edu.unito.it
Processi:

1. Master
2. Atomi
3. Alimentatore
4. Attivatore
5. Alarm
6. Inibitore

Master:
Il processo master ha il compito di:

- creare gli N processi atomi;
- creare il processo Alimentatore;
- creare il processo Attivatore (attraverso il metodo fork() e exvec());
- creare il processo Alarm;
- creare il processo Inibitore se da terminale si è aggiunto il comando “inibitore”;
- impostare i 3 semafori (prioritario e per gli atomi);
- impostare la memoria condivisa (vettore dei pid atomi, numero di atomi, energia ecc…);
- impostare i suoi handler per le terminazioni forzose;
- dare il via ai restanti processi di partire tramite un altro semaforo di sincronizzazione;
- prelevare energia ogni ENERGY_CONSUMPTION secondi;
- stampare le statistiche;
- gestire la terminazione.

Durante l’esecuzione, il processo master dovrà periodicamente mostrare all’utente le statistiche relative agli atomi, accedendo alla memoria condivisa per prelevare una quantità di energia (diminuire quella totale) , leggere e stampare le statistiche.

Per implementare la soluzione, si è pensato di utilizzare un segnale di tipo alarm a se stesso e successivamente accedere alla memoria condivisa attraverso un semaforo prioritario.
Il master usa delle funzioni della libreria master-module.h implementate con il file master-module.c

Atomo:

- Il processo atomo riceve dal processo padre il valore che sarà usato per indicare il numero atomico del processo stesso attraverso fork e cambio di ambiente con execv().
- Durante l'esecuzione(ricevendo un segnale da parte del processo attivatore) dovrà generare una scissione che consiste nel generare un processo figlio che a sua volta darà una parte del suo numero atomico al figlio. Nel caso il **numero atomico sia minore di un determinato valore N_ATOMICO_MIN,** il processo stesso rimuove il suo pid dal vettore dei pid in memoria condivisa, diminuisce il numero degli atomi di 1, somma il suo numero atomico alle scorie e termina. Altrimenti esegue la scissione normalmente;
- Durante la scissione si calcola il numero atomico del figlio e lo si sottrae al proprio;

Anche il processo atomo ha la sua libreria atomo.h
Per aggiungere l'atomo alla memoria condivisa il programma cerca il primo spazio disponibile nell'array già creato, altrimenti crea un nuovo array più grande e ci copia quello vecchio, per poi rimuovere quest'ultimo e collegare quello nuovo.
Nella scissione l'atomo controlla due semafori, quello prioritario e quello degli atomi, perché ha una priorità più bassa rispetto agli altri processi e entrambi i semafori devono essere liberi per far scindere l'atomo.
Viene usata una coda di messaggi per implementare la versione normal con inibitore.

Alimentatore:
Dopo la creazione del processo da parte del master, periodicamente genera un numero costante di processi figli che poi successivamente verranno registrati nel vettore dei processi (memoria condivisa) atomi guardando il semaforo prioritario.

Attivatore:
Dopo la creazione del processo da parte del master, periodicamente controlla il semaforo prioritario, se può accedervi, periodicamente sceglie a caso un processo atomo da scindere.
La scelta dell'atomo vittima viene fatta generando un numero casuale da 0 a **NATOMI** (in mem. condivisa) e prendendo il pid del processo scelto dal vettore dei pid contenuto nella memoria condivisa.
Successivamente l'attivatore manderà a quel processo un segnale di scissione.

Alarm:
Si tratta di un processo creato dal master per gestire il tempo della simulazione. Il programma è composto da un alarm che scatta quando il tempo della simulazione termina, quindi manda un segnale SIGUSR1 al master.

Inibitore:

Il programma usa una coda di messaggi per inviare l'esito probabilistico della scissione e quanta energia sottrarre alla scissione dell'atomo. I messaggi vengono inviati all'atomo che intende scindersi che manda un segnale SIGUSR2 al processo. Il processo può essere messo in pausa in qualsiasi momento mandandogli (da terminale) un segnale di SIGTSTP, per poi essere ripreso con qualsiasi segnale che non lo termini.

Struttura semafori

Per sincronizzare i processi implementiamo due semafori mutex:

- Uno di sincronizzazione, inizialmente -1, posto dal master a 1 per sincronizzare tutti i processi;
- Uno per i processi master, attivatore e alimentatore chiamato **semaforo prioritario**. Questo viene implementato perché si è deciso di dare priorità alle azioni che avvengono ciclicamente per non farle ritardare rispetto alla concorrenza tra atomi. Infatti abbiamo solo 3 processi che vi concorrono per accedere alla memoria condivisa invece che migliaia;
- Uno per i processi atomo, questo semaforo ha meno priorità perché è pensato per far competere tanti processi di cui non si ha fretta di far accedere alla memoria condivisa.

Struttura memoria condivisa

Implementiamo l'uso della memoria condivisa per la grande mole di processi che devono comunicare.

Essa contiene una struct memCond con i seguenti campi:

- l'id del vettore condiviso;
- il numero degli atomi;
- l'energia totale sprigionata;
- il numero delle scissioni eseguite;
- il numero delle attivazioni effettuate;
- l'energia consumata dal reattore;
- le scorie prodotte dal reattore;
- il pid del processo inibitore;
- il numero di scorie prodotte nell'ultimo secondo;
- il numero di attivazioni avvenute nell'ultimo secondo;
- il numero di scissioni avvenute nell'ultimo secondo;
- l'energia del reattore nell'ultimo secondo.

Un'altra struttura è il vettore dei pid, un vettore di interi di lunghezza variabile nAtomi contenente pid degli atomi (interi).

Il vettore dei pid verrà modificato dai processi atomo e dall'alimentatore, così come il numero di atomi e l'energia sprigionata.

Struttura coda di messaggi

La coda di messaggi viene usata dall'inibitore per mandare dei dati all'atomo che vuole scindersi.

Essa è formata da:

- tipo, della coda;
- energia da sottrarre alla scissione;
- esito della scissione, che è 1 nel caso di successo.

risorse.h
Questa libreria contiene tutte le costanti definite per il corretto funzionamento di tutto il programma, contiene anche le definizioni delle due strutture definite precedentemente e la definizione di _GNU_SOURCE.