Mattia Mignogna (matr. 1043330) **Fabio Nebbia** (matr. 898514)

Progetto di Sistemi Operativi

Versione NORMAL

A.A. 2023/2024

Contents

1 Premesse

1.1 Sviluppo

Il progetto è stato sviluppato, compilato e testato su:

	Arch Linux*	**Ubuntu 22.04.3 LTS****	**Ubuntu 23.10****
`gcc`	13.2.1	11.4.0	13.2.0
`gdb`	14.1	12.1	14.0
`make`	4.4.1	4.3 TODO	4.3

* usato per lo sviluppo e il testing.
** usato per il testing.

1.2 Compilazione ed esecuzione del progetto

Tutte le operazioni di controllo della simulazione si effettuano tramite lo script BASH `soctl.sh` presente nella cartella del progetto. Lo script compila automaticamente tutti i moduli del progetto invocando `make` e predispone l'environtment per l'esecuzione delle simulazioni.

Seguono alcuni esempi di utilizzo.

Comando	**Effetto**
`./soctl.sh --help`	Stampa la lista esaustiva dei comandi a disposizione, le relative shortcut e corrispondente sintassi.
`./soctl.sh start --explode --inhibitor`	Carica la configurazione per lo scenario di explode e attiva l'inibitore all'avvio della simulazione.
`./soctl.sh start --meltdown`	Carica la configurazione per lo scenario di meltdown e **non** attiva l'inibitore all'avvio della simulazione.
`./soctl.sh inhibitor toggle`	Attiva o disattiva l'inibitore a simulazione in corso.
`./soctl.sh stop`	Termina manualmente la simulazione in corso.

1.3 Interpretazione della stampa delle statistiche

Ogni secondo il master produrrà a video un output di questo tipo:

```
INFO Inhibitor wasting atom 124205
INFO Inhibitor reducing energy by 114 and wasting atom 124209
INFO Inhibitor reducing energy by 735 and wasting atom 124212

           Global [RUNNING | 5s/60s]
           Total    LastSec
     Atoms 475      9
    Wastes 474      8
  Fissions 252      5
      Acts 474      8

           Inhibitor [ON]
           Total    LastSec
    Wastes 252      5
    Energy 76041    1717

           Total    LastSec  Used     Free
    Energy 118540   2217     27500    14999
```

Figure 1: Stats printed by master

Dove il significato di ogni sezione delineata corrisponde a:

1. Log opzionalmente prodotti dall'inibitore in caso esso sia attivo e il suo log sia abilitato (ossia non sia stato passato a `./soctl.sh start` il flag `--no-inh-log`). Queste indicano, ad ogni scissione, l'energia che è stata assorbita dall'inibitore (se necessario ad evitare `EXPLODE`) e l'atomo che è stato convertito in scoria.
2. Stato globale della simulazione (`RUNNING`, `MELTDOWN`, `EXPLODE`, `TIMEOUT`, `BLACKOUT`, `TERMINATED`), seguito dal numero di secondi restanti alla terminazione per `TIMEOUT` e dalla durata totale della stessa.
3. Riporta il numero di atomi, scorie, fissioni e attivazioni avvenute in totale dall'inizio della simulazione e quelle relative all'ultimo secondo.
4. Stato dell'inibitore (`ON` oppure `OFF`), il quale può essere attivato o disattivato in qualsiasi momento usando gli appositi comandi (`./soctl.sh --help`). Di conseguenza, le relative statistiche (sezione 5), varieranno (o meno) a seconda di quando e come viene manipolato.
5. Riporta il numero di atomi trasformati in scoria dall'inibitore e la quantità di energia assorbita dallo stesso, in totale dall'inizio della simulazione e relativamente all'ultimo secondo.
6. Riporta l'energia prodotta in totale dalla simulazione, quella prodotta nell'ultimo secondo, quella consumata in totale dal master da inizio simulazione e quella correntemente libera.

2 Scelte progettuali

2.1 Programmazione modulare

2.1.1 Suddivisione in moduli e librerie

Il progetto è così genericamente strutturato:

```
project/
  |
  +-- <module>        // tutti i moduli principali (omessi per brevità)
  |
  +-- bin/            // contiene i binari compilati
  |
  +-- env/            // contiene le configurazioni per i vari scenari
  |
  +-- libs/
  |    |
  |    +-- impl/      // implementazioni delle librerie
  |    |
  |    +-- lib/       // header delle librerie
  |
  +-- makefile        // per compilare i moduli e le librerie
  |
  +-- soctl.sh        // per il controllo da terminale della simulazione
```

Ogni processo è implementato in un modulo separato da tutti gli altri e viene così immesso nella simulazione:

- Il master avvia alimentatore, attivatore, inibitore e `N_ATOMI_INIT` tramite `fork` e successiva `execv`;
- L'limentatore crea `N_NUOVI_ATOMI` atomi tramite `fork` e successiva `execv`;
- Diverge l'atomo, che si scinde tramite la sola `fork`.

Alcuni moduli particolari sono:

- `model`, compilato insieme ad ogni modulo principale e fa uso delle direttive del preprocessore per assumere la struttura adeguata per il particolare processo che si sta compilando (sezione 2.1.2 TODO);
- `inhibitor_ctl`, utilizzato tramite `./soctl.sh inhibitor` per controllare lo stato dell'inibitore a run-time.

Sono state realizzate librerie condivise, compilate una sola volta, per la realizzazione delle seguenti funzionalità:

- Interazione con la FIFO (sezione 2.3.1 TODO);
- Interazione con la LIFO (sezione 2.3.2 TODO);
- Interazione con i semafori (sezione 2.4 TODO);
- Signal handling e signal (un)masking (sezione 2.4 TODO);
- Interazione con le memorie condivise;
- Stampa formattata su console;
- Util generiche (math utils, timer, passaggio di argomenti tramite `execv`, file temporanei, ecc).

Sono anche presenti alcuni header non associati a librerie:

- `libs/lib/config.h`, che consente a tutti i processi di accedere facilmente alla configurazione in shared memory;
- `libs/lib/ipc.h`, che contiene informazioni utili ai processi per comunicare tra loro.

2.1.2 Utilizzo di make e makefile

Il `makefile` contiene le opportune direttive per:

- Compilare tutte le librerie;
- Compilre `inhibitor_ctl`, per essere usato tramite `./soctl.sh inhibitor`;
- Compilare i moduli di tutti i principali processi, ciascuno con applicate le opportune differenze in `model`;

Sono state utilizzate diverse funzionalità di `make`, tra cui:

- `%`, per eseguire il matching del nome del modulo che si intende compilare;
- `$@`, `$^`, `$<`, per automatizzare la compilazione senza ripetere i nomi di target/dipendenze;
- `eval` e `shell`, per la define automatica del nome del modulo (es. `-DMASTER`);
- `addprefix`, per abbreviare la stesura del `makefile` stesso;
- `filter`, per selezionare i file corretti da passsare a `gcc`.

Inoltre, per `gcc` sono state utilizzate flag quali:

- `-g`, per eseguire il debugging tramite `gdb`;
- `-I<dir>`, per indicare le directory in cui cercare gli header;
- `-L<dir>`, per indicare la directory in cui il linker può reperire le librerie condivise;
- `-l:<library>`, per indicare i file binari delle singole librerie condivise da linkare.

Consultare direttamente il `makefile` per visionare come sono state impiegate tali funzionalità.

2.2 Configurazione

La configurazione di una simulazione è stata realizzata tramite variabili d'ambiente.

Il master ne effettua la lettura e, accertata la loro correttezza, le insiderisce in memoria condivisa in modo che tutti gli altri processi vi abbiano immediato accesso, senza eseguire a loro volta effettuare letture e parsing numerico.

2.3 Gestione dei pid dei processi atomo

La gestione dei pid è stata ottimizzata allo scopo di massimizzare il numero di scorie prodotte per ridurre al minimo il rischio di `MELTDOWN`.

Per fare questo, dato che il numero atomico è randomico ed è un'informazione privata del processo atomo, ci si è basati sull'euristica per cui un atomo, man mano che viene scisso, vede un progressivo decadimento del suo numero atomico: è più probabile che un atomo scisso abbia numero atomico minore e sia quindi più prossimo al diventare scoria.

Per separare gli atomi "nuovi", ossia quelli che ancora non hanno subito scissioni, da quelli che, invece, si sono scissi più recentemente, si è scelto di usufruire di due strutture dati differenti:

- Quelli "nuovi", immessi dal master e dall'alimentatore, sono memorizzati in una FIFO;
- Quelli scissi dall'attivatore sono memorizzati in una LIFO (implementata in shared memory).
  La natura stessa della struttura dati permette di tenere traccia di quelli scissi più recentemente e, quindi, del progressivo decadimento del relativo numero atomico.

Segue la gestione nei due casi.

2.3.1 FIFO

Gli atomi immessi nella simulazione tramite `fork` (e successiva `execv`) del master e dell'alimentatore memorizzano automaticamente il relativo pid nella FIFO, in quanto è impossibile avere informazioni sul loro numero atomico e si è scelto di processarli dal più vecchio al più nuovo.

2.3.2 LIFO in shared memory

Gli atomi più recentemente scissi dall'attivatore, ammesso che non si traformino in scorie, memorizzano il proprio pid nella LIFO. Quest'ultima risiede in shared memory, in modo tale che sia accessibile a tutti i processi che devono manipolarne lo stato (le manipolazioni effettuate saranno dettagliate in sezione 2.3 TODO.

L'implementazione data, a seconda del fabbisogno determinato dalla configurazione della simulazione, è automaticamente in grado di aumentare (o diminuire) la sua dimensione (richiedendo al SO un nuovo segmento di shared memory delle opportune dimensioni, copiando i dati pre-esistenti e rilasciando il segmento precedente).

2.4 Inibitore e ciclo di attivazione e scissione

Quando un atomo riceve un `SIGACTV` dal processo attivatore, ammesso che non debba trasformarsi in scoria,

2.5 exit in qualsiasi punto senza leak