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

Progetto di Sistemi Operativi

Versione NORMAL

A.A. 2023/2024

Contents

1 Premesse **3**
1.1 Sviluppo . . . 3
1.2 Compilazione ed esecuzione del progetto . . . 3
1.3 Interpretazione della stampa delle statistiche . . . 4

2 Scelte progettuali **5**
2.1 Programmazione modulare . . . 5
2.1.1 Suddivisione in moduli e librerie . . . 5
2.1.2 Utilizzo di make e makefile . . . 6
2.2 Configurazione . . . 6
2.3 Gestione dei pid dei processi atomo . . . 6
2.3.1 FIFO . . . 6
2.3.2 LIFO in shared memory . . . 6
2.4 Processi e lifecycle . . . 7
2.4.1 Master . . . 7
2.4.2 Attivatore . . . 7
2.4.3 Atomo . . . 7
2.4.4 Inibitore . . . 7
2.4.5 Alimentatore . . . 7
2.4.6 Esempio di un ciclo di attivazione e scissione . . . 8

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 la 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 sarà prodotto 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: Log e statistiche stampate a video

Dove il significato di ogni sezione delineata corrisponde a:

1. Log opzionalmente prodotti dall'inibitore in caso 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 (per evitare `MELTDOWN`).
2. Stato globale della simulazione (`RUNNING`, `TIMEOUT`, `EXPLODE`, `BLACKOUT`, `MELTDOWN`, `TERMINATED`), seguito dal numero di secondi restanti alla terminazione per `TIMEOUT` e dalla durata totale configurata per la 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`), quest'ultimo può essere attivato o disattivato in qualsiasi momento usando gli appositi comandi (vedere `./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'alimentatore immette `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, che 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 implementare le 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/prerequisiti;
- `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 delle librerie condivise;
- `-L<dir>`, per indicare la directory in cui il linker può reperire le librerie condivise;
- `-l:<library>`, per indicare al linker i file binari delle singole librerie condivise.

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 inserisce in memoria condivisa in modo che tutti gli altri processi vi abbiano immediato accesso, senza eseguire a loro volta 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 degli atomi scissi più recentemente e, quindi, del progressivo decadimento del relativo numero atomico.

2.3.1 FIFO

Gli atomi immessi nella simulazione dal master e dall'alimentatore memorizzano automaticamente il relativo pid nella FIFO, in quanto è impossibile avere informazioni sul loro numero atomico e si è scelto di processarli in ordine di immissione nella simulazione.

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) lo spazio riservato per la LIFO (richiedendo al SO un nuovo segmento di shared memory delle opportune dimensioni, copiando i dati pre-esistenti e rilasciando il segmento precedente).

2.4 Processi e lifecycle

Nelle seguenti sezioni sono riportate le principali scelte progettuali relative alla "main logic" di tutti i processi.

A simulazione avviata, viene mantenuto uno stato consistente grazie all'uso di semafori, che permettono di regolare in maniera opportuna l'alternarsi delle operazioni dei vari processi.

Un esempio di parte di queste dinamiche è visionabile in sezione 2.4.6 TODO;

Per visionare per esteso la logica di ogni processo e l'esatto utilizzo dei semafori, visionare i sorgenti, opportunamente documentati tramite appositi commenti.

2.4.1 Master

Il master resta in attesa tramite `sigsuspend` di uno dei seguenti segnali:

- `SIGMELT`, tramite cui l'alimentatore o un atomo che ha tentato la scissione comunicano l'avvenuto fallimento di una `fork` e quindi la terminazione per `MELTDOWN`;
- `SIGALRM`, inviato dal `timer` di 1 secondo, inizializzato dal processo stesso, per controllare lo stato della simulazione (ossia terminazioni per `TIMEOUT`, `EXPLODE`, `BLACKOUT`) e stampare le statistiche.
- `SIGTERM`, tramite cui è possibile terminare manualmente la simulazione:
  - da terminale con comando `kill -SIGTERM <master_pid>`;
  - con comando `./soctl.sh stop`, che esegue quanto sopra;

In qualsiasi stato di terminazione (inclusa quella manuale), il master si premura di inviare `SIGTERM` a tutti gli altri processi della simulazione per consentire una graceful exit, per poi procedere al rilascio delle risorse IPC al SO.

2.4.2 Attivatore

L'attivatore resta in attesa tramite `sigsuspend` di uno dei seguenti segnali:

- `SIGALRM`, inviato dal `timer` di `STEP_ATTIVATORE` nanosecondi, inizializzato dal processo stesso, per causare l'attivazione di un atomo, che viene così selezionato:
  - se la LIFO non è vuota, preleva l'atomo scisso più recentemente;
  - se la FIFO non è vuota, preleva il più vecchio atomo inserito;
  - se così facendo è stato selezionato un atomo, gli viene inviato `SIGACTV` per causarne la scissione;
  - se entrambe le strutture sono vuote, non viene selezionato alcun atomo e l'attivazione non avviene.

2.4.3 Atomo

L'atomo resta in attesa tramite `sigsuspend` di uno dei seguenti segnali:

- `SIGACTV`, inviato dall'attivatore, per richiedere la scissione.
  Questo segnale dà il via a una sequenza di operazioni che variano a seconda dello stato dell'atomo e che possono coinvolgere l'inibitore, se presente, per inibire l'energia liberata e trasformare un atomo in scoria;
- `SIGWAST`, inviato dall'inibitore per trasformare un atomo in scoria dopo la scissione;
- `SIGTERM`, inviato dal master per indicare la terminazione della simulazione.

2.4.4 Inibitore

L'inibitore resta in attesa:

- Su un semaforo dedicato che viene incrementato dagli atomi a seguito della scissione, per segnalare all'inibitore di eseguire le relative operazioni di controllo:
  - assorbimento di tutta l'energia necessaria a evitare `EXPLODE`;
  - trasformazione in scoria di uno degli atomi scissi per evitare `MELTDOWN`.
- Di `SIGTERM`, da parte del master, per indicare la terminazione della simulazione.

2.4.5 Alimentatore

L'alimentore esegue le sue funzionalità in due modalità distinte, che dipendono dallo stato dell'inibitore e che, quindi, possono alternarsi un numero arbitrario di volte nel corso di una singola simulazione:

- se l'inibitore è `OFF`, ogni `STEP_ALIMENTAZIONE` esegue la `fork` di `N_NUOVI_ATOMI`, senza limitazioni;
- se l'inibitore è `ON`, ogni `STEP_ALIMENTAZIONE` esegue un numero limitato ($\leq$ `N_NUOVI_ATOMI`) di `fork` indicato dal valore di un apposito semaforo utilizzato per evitare `MELTDOWN`.

2.4.6 Esempio di un ciclo di attivazione e scissione

Di seguito viene riportato un esempio di ciclo di attivazione, in particolare uno in cui sia presente il processo inibitore:

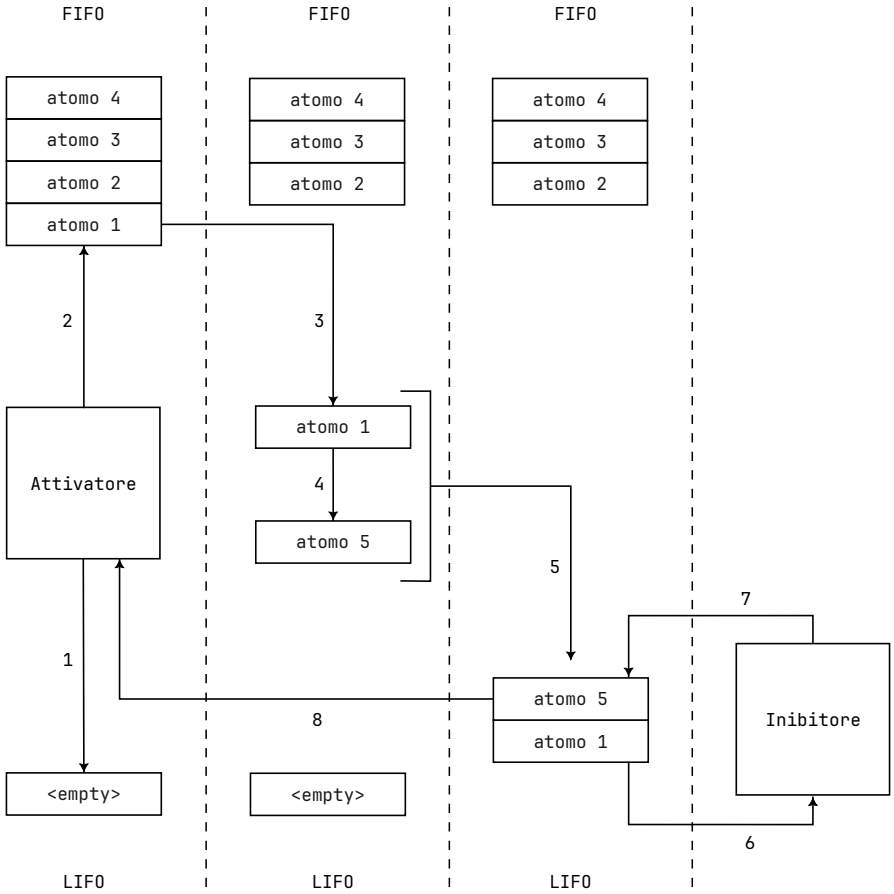

Figure 2: Esempio di un ipotetico ciclo di attivazione

Dove le frecce indicano le seguenti operazioni:

1. L'attivatore tenta per prima cosa di rimuovere dalla LIFO un atomo scisso di recente, tuttavia la LIFO è vuota.
2. L'attivatore ne rimuove, quindi, uno dalla FIFO.
3. L'attivatore provvede all'invio di `SIGACTV` all'atomo selezionato.
4. L'atomo (padre) si scinde, cioè crea un nuovo atomo (figlio), produce energia e aggiorna le statistiche.
5. L'atomo padre inserisce in LIFO (atomi scissi recentemente) se stesso e l'atomo figlio appena generato.
6. L'atomo padre incrementa il semaforo dell'inibitore per permettergli di svolgere le sue funzioni.
7. L'inibitore trasforma in scoria l'atomo figlio tramite l'invio di `SIGWAST`.
8. L'atomo figlio aggiorna le statistiche e, solo dopo che l'inibitore ha terminato, restituisce il controllo al master (in caso debba eseguire le relative operazioni di controllo) e successivamente di nuovo all'attivatore (per procedere con un nuovo ciclo di attivazione). Infine, l'atomo figlio termina la sua esecuzione.