Relazione Progetto

Studente: Sacchetti Lorenzo – 1043821

Insegnamento: Sistemi Operativi

Struttura del progetto:

Il progetto è suddiviso nelle seguenti parti:

(contenuto del file README)

- Makefile
- README
- bin <- Project main executable
- build <- Static object and intermediate file
- config <- Gonfiguration file for the simulation
- docs <- Documentation
- include <- Header files
- src <- Source files
- tmp <- Temporary file generated by the program

All'interno delle cartelle "src" e "include" sono presenti i file *.c e i rispettivi header *.h.

Nelle cartelle "build" e "bin" sono presenti i risultati della compilazione e del linking quindi i file eseguibili.

Nella cartella "config" sono presenti i file *.txt con le configurazioni. Ogni file presente contiene una configurazione separata.

La compilazione avviene tramite l'uso della make utility in fasi separate. Prima la compilazione con la creazione dei file oggetto, in seguito il linking per la creazione degli eseguibili.

Funzionamento:

Il progetto viene lanciato tramite l'eseguibile "master.out" che avvia tutti gli altri elementi di controllo della simulazione e, tramite l'uso di un apposito set di semafori, la avvia in accordo con lo stato di prontezza dei vari processi.

Ogni processo all'avvio esegue una funzione di init che inizializza tutte le strutture dati necessarie come timer e IPC.

Tutti i processi eseguono le loro operazioni ti routine tramite la gestione di segnali lanciati ogni quanto di tempo da appositi timer. Nello specifico il master esegue una stampa delle statistiche ogni secondo, alimentazione genera nuovi atomi ogni STEP secondi, attivatore comanda ACTIVATION_PER_SECOND split degli atomi ogni secondo.

La comunicazione di informazioni tra processi avviene principalmente con l'uno di code di messaggi e memoria condivisa. La memoria condivisa è protetta da un semaforo dedicato.

Processo attivatore: questo processo sceglie in maniera casuale uno tra gli atomi disponibili (che non sia una scoria) e, tramite un segnale, lo attiva.

Processo atomo: a seguito della fase di init il processo rimane in pausa fino a quando non riceve un segnale di attivazione. A seguito dei dovuti controlli sul suo stato avvia la fase di scissione eseguendo una richiesta

preventiva al processo inibitore. La richiesta è gestita tramite coda di messaggi. Se il processo inibitore concede la scissione, viene generato un altro atomo e il numero atomico opportunamente suddiviso e comunicato tramite coda di messaggi. L'aumento di energia previsto viene salvato in memoria condivisa. Terminate le operazioni l'atomo ritorna in pausa.

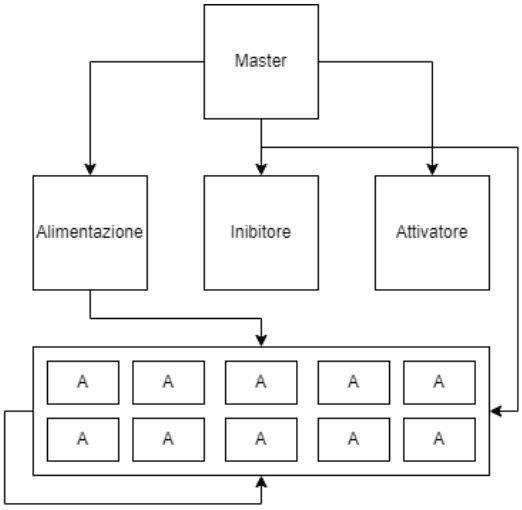
Processo inibitore: a seguito della fase di init il processo rimane il attesa di ricevere un messaggio da parte di un atomo che richiede di dividersi. Vengono eseguiti i controlli sulle possibili emissioni energetiche e, a seguito dei controlli di tutte le soglie limite, il processo decide di concedere oppure no il permesso all'atomo di dividersi. Tutte le operazioni di comunicazioni avvengono tramite coda di messaggi. I valori energetici complessivi sono mantenuti in memoria condivisa.

Processo alimentazione: genera nuovi atomi eseguendo dei fork e richiamando l'eseguibile per l'atomo tramite execve.

Processo master: questo processo durante la fase di init genera tutte le strutture di IPC che in seguito gli altri processi andranno ad utilizzare. Ogni secondo esegue in controlli sui valori della simulazione e se necessario termina in uno dei casi previsti. Ogni secondo stampa i valori parziali e totali della simulazione. Il processo master è l'unico processo a gestire la creazione dei canali di IPC sia in fase di inizializzazione che al termine della simulazione. Al termine della simulazione dopo aver aspettato la terminazione di tutti i processi, chiude i canali di IPC e salva su file la situazione complessiva della simulazione insieme allo stato corrente della memoria condivisa.

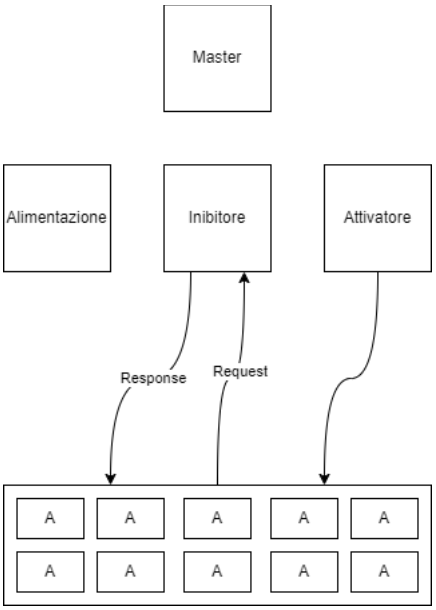
Fase di creazione dei processi.

I processi vengono creati utilizzando la funzione fork seguita da un'apposita funzione execve. Inizialmente il processo master crea tutti gli elementi per la simulazione. In seguito durante la simulazione gli atomi vengono generati rispettivamente dal processo alimentazione o dagli atomi stessi. Il numero atomico viene comunicato agli atomi tramite una coda di messaggi. Il valore è estratto secondo una distribuzione di probabilità normale che genera numeri nell'intorno di N_ATOM_MAX/2. (nella cartella docs è presente il file chart.svg che dimostra che l'estrazione dei numeri avviene secondo una curva normale). Durante la fase di scissione gli atomi oltre ad eseguire una fork eseguono anche execve in modo da mantenere uniforme la tecnica di creazione di nuovi processi.



Fase di scissione.

Casualmente il processo attivatore cerca in memoria condivisa un processo valido per la scissione. Tramite un segnale gli comunica l'ordine di eseguire tale operazione. Il processo atomo inoltra una richiesta al processo inibitore che dopo aver effettuato tutte le verifiche di rito decide se consentire la scissione oppure no. La comunicazione tra atomo e inibitore avviene tramite coda di messaggi inviando un'istanza di una struttura dati creata appositamente per condividere le informazioni necessarie alla scissione come pin e numero atomico. In totale la scissione avviene in 3 fasi indicate dalle frecce nello schema sottostante.



Gestione dell'attesa

I processi devono assolutamente evitare l'attesa attiva e questo avviene principalmente in due modi.

I processi master, alimentazione, atomo, e attivatore al termine delle operazioni di init eseguono pause() che manda il processo in sleep fino a quando un segnale non viene ricevuto. I processi che periodicamente devono svolgere delle operazioni sfruttano un timer che invia a se stesso un segnale usando al funzione raise(). Il segnale viene intercettato, il lavoro viene svolto e al termine il processo riesegue pause().

Il processo inibitore funziona diversamente. Poiché non è possibile controllare in continuazione se un messaggio e arrivato in coda si sfrutta la capacità della funzione msgrcv() di far attendere il processo fino alla ricezione di un messaggio. Quando il messaggio arriva in coda viene gestito e inoltrata la risposta ad atomo che nel frattempo ha lo stesso comportamento (solo per la fase di richiesta di scissione).

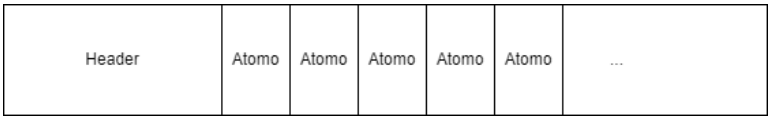
```
  0[|||                                    3.6%]     4[||                                    2.4%]
  1[||                                     3.0%]     5[|||                                   3.0%]
  2[||                                     2.4%]     6[||                                    2.4%]
  3[|||||||                               10.8%]     7[||                                    2.4%]
Mem[||||||||||||||||||                1.45G/7.71G]   Tasks: 260, 319 thr; 1 running
Swp[                                    0K/3.14G]   Load average: 0.45 0.36 0.27
                                                    Uptime: 07:38:16

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  17723 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17722 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17721 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17720 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17719 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17718 lorenzo    20   0 6453M  564M  163M S  0.0  7.1  0:00.00 /usr/bin/gnome-shell
  17717 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17716 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17715 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17714 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17713 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17706 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17705 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
  17704 lorenzo    20   0  4312  1920  1920 S  0.0  0.0  0:00.00 ./atomo.out ../config/config_0.txt
```

utilizzo delle cpu con molti processi attivi

Gestione della comunicazione tra processi

La condivisione di informazioni tra processi avviene tramite code di messaggi e memoria condivisa. Come scelta implementativa è stato deciso di mantenere in memoria condivisa alcuni dati sugli atomi in modo da avere una panoramica di quello che sta accadendo nella simulazione.

La memoria condivisa è strutturata con un header che mantiene le informazioni generali della comunicazione e con un'area di memoria dedicata ad ogni atomo. Viene usata una sola area di memoria.



Per la comunicazione tramite coda di messaggi è stato sviluppato un sistema per permettere la comunicazione bidirezionale da molti processi a uno solo e viceversa da un processo verso molti. Si prende per esempio la comunicazione tra gli atomi e l'inibitore.

Tutti gli atomi che devono inoltrare una richiesta all'inibitore inseriscono nel corpo del messaggio il proprio pid ma mantengono il parametro mtype constante (1 in questo caso). Il processo inibitore legge tutti i messaggi che hanno come mtype il valore costante designato e utilizza il pid presente nel corpo del messaggio come mtype per la risposta. L'atomo che nel mentre rimane in attesa, come già descritto nel paragrafo sulla gestione dell'attesa, legge i messaggi che hanno come mtype il valore del proprio pid. In questo modo è possibile isolare la comunicazione e di fatto renderla uno a uno sfruttando però una sola coda. Questo metodo evita la creazione di un grad numero di code e di conseguenza la necessità di praticare un gran numero di controlli su ognuna di esse.

Console

Per la gestione della simulazione e in particolar modo del processo inibitore è presente una console di controllo che, tramite appositi comandi consente di terminare la simulazione, attivare o disattivare il processo inibitore. La simulazione può funzionare anche senza la console ma il processo inibitore verrà attivato in accordo con il valore presente nel file di configurazione.

Dettagli sul codice

Funzione init

Questa funzione è presente in tutti gli eseguibili. La funzione inizializza le variabili locali, accedere ai canali IPC e legge il file di configurazione. La funzione non è definita in un file comune poiché ogni processo necessità di personalizzazioni a seconda del compito che deve svolgere, lo stesso vale per la lettura del file di configurazione che varia da processo a processo. Tra le operazioni svolte è possibile trovare l'inizializzazione di semafori, code di messaggi, memoria condivisa, handler dei segnali che vengono seguite dall'inizializzazione dei timer dove previsto e dalla lettura del file di configurazione. Questa funzione nel processo master è più specializzata poiché è l'unica che può effettivamente richiedere i canali IPC.

Funzione handle_signals

Consente la gestione dei segnali intercettati. I segnali sono principalmente due ovvero SIGINT E SIGUSR1.

Quando viene intercettato SIGINT il processo master invia a tutti processi un segnale SIGINT che viene intercettato e interpretato come segnale di terminazione del processo. Successivamente vengono chiusi tutti i canali di IPC ma non prima di aver eseguito un salvataggio su file dello stato della memoria alla terminazione della simulazione.

Il segnale SIGUSR1 produce un risultato diverso a seconda del processo che lo intercetta. Il master esegue la stampa delle statistiche e controlla se i dati sono conformi alle specifiche. In caso di anomalia produce uno dei risultati previsti (explode, blackout). Il processo alimentazioni genera nuovi processi atomo, il processo inibitore sottrae energia, il processo attivatore seleziona gli atomi da scindere e il processo atomo avvia la procedura di scissione.

Funzione normalDistributionNumberGenerator

Genera un numero secondo una distribuzione normale. Sfrutta la tecnica Box-Muller che, inseriti due numeri random ne genera uno solo. Testando la tecnica su un numero significativo di operazioni è possibile confermare che i numeri vengono generati secondo una distribuzione normale.

Funzione getValueFromConfigFile

Legge il file di configurazione per ottenere le costanti di controllo della simulazione. La funzione e ridefinita per ogni processo con le costanti necessarie. Nel processo master viene inoltre eseguito il comando ulimit -a con reindirizzamento dell'output su un file, questo è necessario a sapere il limite di processi utente da non sforare.

Funzione Write

È sostanzialmente un wrapper della originale funzione write. Esegue la stampa di colore diverso a seconda del processo.

Sviluppo

lo sviluppo e il testing e avvenuto in accordo con le specifiche ANSI su sistema operativi Ubuntu 22.04.3 LTS. La shell utilizzata per lo sviluppo è GNU bash versione 5.1.16(1)-release

Non sono state utilizzate funzioni di libreria ti terze parti. Tutte le definizioni delle funzioni usate sono state prese dalle dispense rese disponibili durante il corso e dal manuale del sistema operativo.

Il codice e il funzionamento dei programmi sono stati testati utilizzando gli strumenti watch e htop.

Il particolare il comando watch -n 1 ipcs fornisce una panoramica nei canali IPC ogni secondo, htop mostra i processi e le loro caratteristiche in forma di tabella aggiornata ogni secondo.