Relazione Progetto S.O.

Anno Accademico 2023/2024

Indice:

Partecipanti:

Nome e Cognome	Matricola	Turno
Andreas Maher Ali Apetrii	1060242	T2
Luca Cavallotto	1060412	T2
Alessandro Demo	1049825	T1

Descrizione processi:

Master

Il processo Master gestisce l'intera **simulazione** e crea **processi figli**.
Lo stato corrente della simulazione e i dati ricavati vengono stampati ogni secondo per osservare il corretto funzionamento di ogni processo.
Inoltre, il processo Master crea **N_ATOMI_INIT** come processi Atomo, il processo Attivatore e il processo Alimentazione.
Si occupa anche delle varie terminazioni e della deallocazione della memoria condivisa, semafori.

Atomo

Il processo Atomo ha un numero atomico generato in **maniera casuale** (un numero compreso tra 1 e **N_ATOM_MAX,** valore preso in input).
Un processo atomo quando esegue la scissione si occupa di assegnare opportunamente al nuovo atomo un numero atomico, e di aggiornare il suo.
Per simulare la **scissione** di ogni atomo si fa l'uso della **fork()**.

Scissione

L'Attivatore avvisa l'atomo di scindersi tramite il segnale SIGUSR1, gestito dalla funzione **signal_handler_sciss()**. Questo metodo, come prima cosa, verifica di avere il numero atomico maggiore di **MIN_N_ATOMICO**, se è minore o uguale creerà una scoria, mentre se è maggiore effettuerà la scissione, estraendo un numero casuale che verrà assegnato come numero atomico del processo figlio, mentre al processo padre verrà sottratto quel numero per ottenere il suo nuovo numero atomico.

Alimentazione

Ogni **STEP** nanosecondi, il processo Alimentazione immette del nuovo combustibile, ovvero crea nuovi atomi, che vengono assegnati alla variabile **N_NUOVI_ATOMI**.

STEP e **N_NUOVI_ATOMI** vengono letti a tempo di esecuzione dal file ‘config.txt’.

Attivatore

Il compito del processo Attivatore è quello di ordinare ad un atomo, scelto in maniera randomica tra quelli disponibili (vedere **Gestione Pipe**), di **scindersi**, inoltre si occupa parzialmente della **terminazione per timeout**, tramite l’utilizzo del metodo **alarm_handler()**, il quale comunica al Master di terminare.

Come funziona la scissione

Ogni secondo avviene una scissione, la procedura parte dal processo Master, che nel metodo **alarm_handler()** invia il segnale **SIGUSR2** al processo attivatore, quest’ultimo nell’handler di **SIGUSR2** invierà all’indice che si trova nella **pipe** (per approfondimenti guardare **‘Gestione Pipe’**) il segnale **SIGUSR1**, infine nel processo atomo, ricevuto questo segnale se possibile avverrà la scissione, altrimenti questo processo diventerà scoria per poi terminare.

Gestione Segnali:

Nome Segnale	Inviante	Ricevente Descrizione
SIGTERM	Alimentazione Atomo	Master, quando una qualsiasi fork fallisce viene inviato questo segnale per la terminazione **meltdown**.
SIGALRM	Master	Master, quando viene ricevuto il segnale di **alarm** il master scrive in output nella console la **stampa 'giornaliera'**.
SIGUSR2	Attivatore	Master, quando ricevuto questo segnale da parte del processo Attivatore si termina la **simulazione per 'timeout'**.
SIGALRM	Attivatore	Attivatore, se ricevuto indica che il tempo timeout della simulazione è scaduto, invia quindi al processo Master il segnale **SIGUSR2**.
SIGUSR2	Master	Attivatore, quando viene ricevuto dal processo Master inizia ad inviare segnali all'atomo con indice n per ordinare loro di **scindersi**.
SIGUSR1	Attivatore	Atomo, quando viene ricevuto l'atomo se possibile si scinde, altrimenti diventa **scoria**.
SIGALARM	Attivatore	Attivatore, dopo step nanosecondi il processo Alimentazione crea **nuovi atomi**.

Gestione Memoria Condivisa:

Per la realizzazione di questo progetto si utilizza una memoria condivisa con dentro una struct chiamata **'atomo'**. La memoria condivisa viene utilizzata da tutti i processi per la gestione delle statistiche e il salvataggio dei valori della configurazione per inizializzare la reazione a catena.

Di seguito il codice di **'struct.h'**:

```
struct atomo{
    pid_t master;
    double energy_produced;
    double energy_consumed;
    double energy_rel_prod;
    double energy_rel_con;
    int atomo[1000];
    int n_atom[1000];
    int curr_idx;
    int scorie;
    int attivazioni;
    int attivazioni_rel;
    int scissioni;
    int scissioni_rel;
    int MIN_N_ATOMICO;
    int MAX_N_ATOMICO;
    int N_NUOVI_ATOMI;
    int STEP;
    int SIM_DURATION;
    int N_ATOM_AT_ONCE;
    int N_ATOMI_INIT;
    float ENERGY_EXPLODE_THRESHOLD;
};
```

Gestione Inizializzazione:

Il metodo di inizializzazione viene gestito nel processo Master tramite la lettura dei valori da assegnare a **tempo di esecuzione** presenti nel file 'config.txt'.
Dopo che la lettura dei valori è stata effettuata, i valori verranno assegnati, tramite una comparazione tra stringhe, alle variabili prestabilite. Tutto ciò è visibile grazie a una stampa finale di ogni variabile e del proprio valore assegnatogli.
Finita l'inizializzazione delle variabili assegnabili a tempo di esecuzione, vengono gestite tutte le variabili restanti, a cui vengono assegnati valori massimi o minimi.

Qui di seguito è presente l'intero metodo **'void init()'**:

```
void init() {
    FILE *oc = fopen("config.txt", "r");
    char buffer[128];
    char c;
    if(oc == NULL){
       printf("Errore nell'apertura del file. \n");
    }
    else{
       printf("Contenuto della configurazione: \n");
       while(fscanf(oc, "%s %c", buffer, &c) != EOF){
           if(strcmp(buffer, "MIN_N_ATOMICO") == 0){
              fscanf(oc, "%d %c", &my_atoms->MIN_N_ATOMICO, &c);
           }
           if(strcmp(buffer, "MAX_N_ATOMICO") == 0){
              fscanf(oc, "%d %c", &my_atoms->MAX_N_ATOMICO, &c);
           }
           if(strcmp(buffer, "STEP") == 0){
              fscanf(oc, "%d %c", &my_atoms->STEP, &c);
           }
           if(strcmp(buffer, "SIM_DURATION") == 0){
              fscanf(oc, "%d %c", &my_atoms->SIM_DURATION, &c);
           }
           if(strcmp(buffer, "N_ATOM_AT_ONCE") == 0){
              fscanf(oc, "%d %c", &my_atoms->N_ATOM_AT_ONCE, &c);
           }
           if(strcmp(buffer, "N_ATOMI_INIT") == 0){
              fscanf(oc, "%d %c", &my_atoms->N_ATOMI_INIT, &c);
```

```
            }
            if(strcmp(buffer, "NUM_CHILDREN") == 0){
               fscanf(oc, "%d %c", &NUM_CHILDREN, &c);
            }
            if(strcmp(buffer, "ENERGY_EXPLODE_THRESHOLD") == 0){
               fscanf(oc, "%f %c", &my_atoms->ENERGY_EXPLODE_THRESHOLD,
&c);
            }
        }
    }
    fclose(oc);
    printf("MIN_N_ATOMICO = %d; \n", my_atoms->MIN_N_ATOMICO);
    printf("MAX_N_ATOMICO = %d; \n", my_atoms->MAX_N_ATOMICO);
    printf("STEP = %d; \n", my_atoms->STEP);
    printf("SIM_DURATION = %d; \n", my_atoms->SIM_DURATION);
    printf("N_ATOM_AT_ONCE = %d; \n", my_atoms->N_ATOM_AT_ONCE);
    printf("N_ATOMI_INIT = %d; \n", my_atoms->N_ATOMI_INIT);
    printf("NUM_CHILDREN = %d; \n", NUM_CHILDREN);
    printf("ENERGY_EXPLODE_THRESHOLD = %f; \n",
my_atoms->ENERGY_EXPLODE_THRESHOLD);
    my_atoms->master = getpid();
    my_atoms->scissioni = 0;
    my_atoms->attivazioni = 0;
    my_atoms->attivazioni_rel = 0;
    my_atoms->scissioni_rel = 0;
    my_atoms->energy_produced = 100;
    my_atoms->energy_consumed = 0;
    my_atoms->energy_rel_prod = 0;
    my_atoms->energy_rel_con = 0;
    my_atoms->scorie = 0;
    N_ATOMI_INIT = my_atoms->N_ATOMI_INIT;
    ENERGY_EXPLODE_THRESHOLD = my_atoms->ENERGY_EXPLODE_THRESHOLD;

}
```

Gestione Pipe:

La pipe implementata serve a creare un canale di comunicazione tra il processo Master (**sender**) e il processo Attivatore (**receiver**).
Attraverso l'utilizzo della pipe si comunica al processo Attivatore l'indice dove trovare il pid al quale dovrà ordinare la scissione.

Gestione Terminazioni:

Le 4 terminazioni **meltdown, blackout, explode** e **timeout** sono tutte configurabili a tempo di esecuzione, modificando opportunamente i valori all'interno del file 'config.txt'.

Le terminazioni sono interamente gestite dal processo Master tramite i seguenti metodi :

- **terminazione_timeout();**
- **terminazione_meltdown();**
- **terminazione_explode();**
- **terminazione_blackout();**