**Descrizione del Progetto**

Il progetto si compone di vari processi che cooperano per gestire la creazione, la gestione e la terminazione di "atomi". I processi principali sono master, atomo, attivatore, alimentatore e inibitore.

**Processo master**

Il processo master è il punto di partenza del programma. Si occupa di:

1. **Creazione di IPCS**: Inizializza le strutture di comunicazione interprocessuali leggendo un file di configurazione (sim.conf).
2. **Inizializzazione dei Semafori**: Imposta diversi semafori (sem\_shm, sem\_activator, sem\_fission, sem\_processes, sem\_inhibitor) con un valore iniziale di 1.
3. **Creazione di Memoria Condivisa**: Attiva e associa segmenti di memoria condivisa per memorizzare variabili globali e stati del sistema.
4. **Gestione dei Segnali**: Configura la gestione dei segnali per gestire interruzioni (ad esempio, SIGINT).
5. **Creazione di Processi Figli**: Crea i processi figli (attivatore, alimentatore, inibitore e atomo) usando fork().
6. **Esecuzione processi**: La funzione execl() si occupa di eseguire i processi creati.

**Processo attivatore.c**

Il processo attivatore invia messaggi agli atomi:

Il processo attivatore è strutturato come un ciclo che continua fin quando il exit\_flag non é attivo e dorme per una quantità di tempo pari a STEP\_ATTIVATORE ogni singola iterazione. Si occupa di inviare messaggi tramite la coda di messaggi inizializzata precedentemente pari al numero di N\_MSG. Ogni singolo messaggio, questo, identifica un comando di fissione per l’atomo.

**Processo alimentatore.c**

Il processo alimentatore crea nuovi atomi periodicamente:

Il processo alimentatore è strutturato in maniera simile ad attivatore; quindi, con un ciclo che continua fin quando il exit\_flag è diverso da 1. Il tutto è regolato da una nanosleep ( ) che addormenta il processo per STEP\_ALIMENTAZIONE nanosecondi. Ad ogni iterazione vengono creati una quantità pari a N\_NUOVI\_ATOMI e vengono immessi dentro la nostra centrale.

**Processo atomo.c**

Il processo atomo rappresenta un atomo nel sistema:

Il processo atomo gestisce la creazione e la divisione degli atomi all'interno del sistema. Utilizza la comunicazione attraverso una coda di messaggi per ricevere i messaggi inviati dall’attivatore. Quando riceve un messaggio, estrae un atomo dalla memoria condivisa e crea un nuovo processo figlio per eseguire l'operazione di "scissione". Dopo ogni operazione, il processo atomo aggiorna le strutture condivise e continua a eseguire finché non riceve un segnale di terminazione. Prima della terminazione esegue periodicamente una stampa di riepilogo sullo stato attuale della simulazione chiamando la sim\_overview(). Una volta terminata l'esecuzione, il processo atomo rilascia le risorse IPC e termina.

La funzione do\_fission() è progettata per simulare il processo di scissione nucleare di un atomo.

**Funzione do\_fission ( )**

Prende in ingresso un atomo (atom\_parent) che verrà sottoposto al processo di scissione.

Se il numero atomico dell'atomo è adeguato: Viene generato un nuovo atomo figlio per simulare la scissione reale degli atomi. Una volta eseguito viene liberata l’energia prodotta dalla scissione e poi immagazzinata nella nostra centrale.

La funzione presenta i controlli necessari affinché la centrale lavori in maniera ottimale con la presenza dell’inibitore attivo ed eventuali controlli per gestire lo stato attuale della gestione della centrale in caso l’energia prodotta superasse il limite massimo disponibile pari a ENERGY\_EXPLODE\_THRESHOLD.

**Processo inibitore.c**

Il processo inibitore serve per sincronizzare e limitare alcune operazioni:

Si fonda su un ciclo infinito che ci permette di attivare e disattivare le funzionalità dell’inibitore a run-time tramite la implementazione dei sig\_handler che ricevono input da tastiera (CTRL + C) regolato da una nanosleep ( ) che lo addormenta per STEP\_INHIBITOR nanosecondi finché non riceve un segnale di terminazione.

**Strutture IPC Utilizzate**

1. **Memoria Condivisa (Shared Memory)**:
   * Utilizzata per memorizzare variabili globali e lo stato del sistema (power\_plant, vars,inhibitor,atoms).
   * Accesso sincronizzato con semafori per evitare condizioni di gara.
2. **Semafori (Semaphores)**:
   * Usati per controllare l'accesso alla memoria condivisa e per sincronizzare i processi (sem\_shm, sem\_activator, sem\_fission, sem\_processes, sem\_inhibitor).
3. **Code di Messaggi (Message Queues)**:
   * Utilizzate per la comunicazione tra processi, in particolare per inviare messaggi agli atomi (msg\_stack).

**Considerazioni Implementative**

* **Sincronizzazione**: L'uso di semafori garantisce che solo un processo alla volta possa accedere a sezioni critiche della memoria condivisa.
* **Comunicazione**: Le code di messaggi permettono una comunicazione efficiente e asincrona tra i processi.
* **Gestione dei Segnali**: La gestione dei segnali consente una chiusura ordinata e reattiva del sistema in caso di interruzioni esterne.