# Corso di Sistemi Operativi, Facoltà di Informatica - Università di Torino

Relazione Progetto Sistemi Operativi
Anno Accademico 2023/2024

Siviero Francesco
Grillo Giovanni
Olivero Alessandro

Maggio 2024

## 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
- Inibitore

## Processo Master

Il processo master è il punto di partenza del programma e si occupa delle seguenti attività:

1. **Creazione di IPCS**: Inizializza le strutture di comunicazione interprocessuali leggendo un file di configurazione (`Progetto.conf`).
2. **Inizializzazione dei Semafori**: Imposta diversi semafori (`semShm`, `semAttivatore`, `semFissione`, `semProcessi`, `semInibitore`) 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` e `execl`.

## Processo Attivatore

Il processo attivatore invia messaggi agli atomi. È strutturato come un ciclo che continua finché il `flagTerminazione` non è attivo e dorme per una quantità di tempo pari a `STEP_ATTIVATORE` ad ogni iterazione. Si occupa di inviare messaggi tramite la coda di messaggi inizializzata precedentemente, per un numero di `N_MSG`. Ogni messaggio identifica un comando di fissione per l'atomo.

## Processo Alimentatore

Il processo alimentatore crea nuovi atomi periodicamente. È strutturato in maniera simile al processo attivatore, con un ciclo che continua finché il `flagTerminazione` è diverso da 1. La regolazione del ciclo è effettuata tramite `nanosleep` che addormenta il processo per `STEP_ALIMENTAZIONE` nanosecondi. Ad ogni iterazione vengono creati una quantità pari a `N_NUOVI_ATOMI` e vengono immessi nella centrale.

## Processo Atomo

Il processo atomo rappresenta un atomo nel sistema. 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. Una volta terminata l'esecuzione, rilascia le risorse IPC e si chiude.

### Funzione `esegui_scissione`

La funzione `esegui_scissione` è progettata per simulare il processo di scissione nucleare di un atomo.

- Prende in ingresso un atomo (`a_PADRE`) 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.
- Viene liberata l'energia prodotta dalla scissione e poi immagazzinata nella 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

Il processo inibitore serve per sincronizzare e limitare alcune operazioni. Si fonda su un ciclo infinito che permette di attivare e disattivare le funzionalità dell'inibitore a run-time tramite l'implementazione dei gestori di segnale (`sig_handler`) che ricevono input da tastiera (CTRL + C) 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 (`centrale`, `var`, `inibitore`, `atomo`).
   - Accesso sincronizzato con semafori.
2. **Semafori (Semaphores)**
   - Usati per controllare l'accesso alla memoria condivisa e per sincronizzare i processi (`semShm`, `semAttivatore`, `semFissione`, `semProcessi`, `semInibitore`).
3. **Code di Messaggi (Message Queues)**
   - Utilizzate per la comunicazione tra processi, in particolare per inviare messaggi agli atomi (`msgPila`).

## Scelte 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.