UNIVERSITÀ
DI TORINO

Laboratorio Sistemi Operativi 2023/2024

Relazione del progetto di:

Caricasulo Francesco Matr. 944173 francesco.caricasulo@edu.unito.it
Persico Loris Matr. 949352 loris.persico@edu.unito.it
Rinaldi Falvio Matr. 955750 flavio.rinaldi@edu.unito.it

Richiesta:

Si intende simulare una reazione a catena. A tal fine sono presenti i seguenti processi:

- Un processo *master* che gestisce la simulazione e mantiene le statistiche;

- Processi *atomo* che si scindono in altri processi atomo, generando energia;

- Un processo *attivatore*;

- Un processo *alimentatore*;

Scelte Implementative:

PROCESSO MASTER

Il processo master viene utilizzato inizialmente per l'avvio dei processi iniziali (atomo, attivatore e alimentatore) e successivamente per il monitoraggio della simulazione stampando a video le statistiche delle varie operazioni.

Il processo Master riesce ad accedere ai dati che riguardano le statistiche della simulazione tramite l'utilizzo della *memoria condivisa*. Nella fase di inizializzazione del processo master, vengono generati l'id della memoria condivisa e l'array ad essa collegato tramite le operazioni:

```
// Ottieni l'ID della memoria condivisa
    shmid = shmget(SHM_KEY, SIZE_STATISTICHE * sizeof(int), IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("Errore nella creazione della memoria condivisa");
        exit(EXIT_FAILURE);
    }

    // Attacca la memoria condivisa al processo
    shared_array = (int *)shmat(shmid, NULL, 0);
    if (shared_array == (int *)(-1)) {
        perror("Errore nell'attaccare la memoria condivisa");
        exit(EXIT_FAILURE);
    }
```

Successivamente, prima di accedere alla memoria condivisa richiede il *semaforo* mediante l'operazione:

```
// Acquisisci il semaforo
if (semop(semid, &acquire_operation, 1) == -1) {
    perror("Errore nell'acquisizione del semaforo");
}
```

Una volta ottenuto il semaforo, siamo in grado di accedere in maniera sicura alla memoria condivisa; perciò, verifichiamo che non siano state raggiunte le opzioni di chiusura della simulazione e successivamente mostriamo a video i dati delle operazioni.

PROCESSO ATOMO:

Il processo atomo viene utilizzato per la creazione dei vari processi atomo successivi all'inizializzazione e per la gestione dei vari numeri atomici e l'energia liberata da queste operazioni.

L'atomo una volta creato si collega alla *coda di messaggi lato scrittura* e invia il proprio ProcessID all'attivatore mediante le seguenti operazioni:

```
// Invio del PID all'Attivatore tramite coda di messaggi
    struct {
        long msg_type;
        pid_t pid;
    } msg;

    msg.msg_type = 1; // Utilizziamo 1 come tipo di messaggio
    msg.pid = pid;

    if (msgsnd(msg_id_1, &msg, sizeof(msg.pid), 0) == -1) {
        perror("Errore nell'invio del PID all'Attivatore");
        exit(EXIT_FAILURE);
    }
```

E poi si mette in attesa di ricevere un messaggio dall'attivatore per poter eseguire l'operazione di scissione. Questa operazione viene eseguita aggiornando inizialmente l'array della memoria condivisa con i nuovi dati sulla scissione e sull'energia liberata calcolata tramite l'operazione:

```
int energy(int n1, int n2) {
    return (n1 * n2) - max(n1, n2);
}
```

(n1 = numero atomico del padre, n2 = numero atomico del figlio)

e successivamente eseguendo una fork, inizializzando il nuovo processo atomo tramite un numero atomico generato dalla funzione:

```
int generate_atomic_number(int nAtomoP) {
    srand(time(NULL));
    return (rand() % nAtomoP);
}
```

Una volta eseguito ciò il processo si rimette in attesa di un successivo messaggio da parte dell'attivatore.

PROCESSO ATTIVATORE:

Il processo Attivatore si occupa di gestire le scissioni da parte dei processi atomo, come prima cosa l'attivatore si collega alla *coda di messaggi lato lettura* e alla memoria condivisa.

Successivamente, ottiene tutti i messaggi presenti sulla coda di messaggi e ne seleziona uno al quale mandare il messaggio di scissione mediante le operazioni:

```
struct msqid_ds queue_info;
        if (msgctl(msg_id_1, IPC_STAT, &queue_info) == -1) {
            perror("Errore nel recupero delle informazioni sulla coda di
messaggi");
            exit(EXIT_FAILURE);
        }
------
if ((msgrcv(msg_id_1, &message, sizeof(message.pid), 1, IPC_NOWAIT) == -1)) {
                //perror("Errore nella ricezione del messaggio dalla coda di
messaggi");
                printf("Tipo del messaggio ricevuto: %d\n", message.msg_type);
                //exit(EXIT_FAILURE);
            }
------
 pid_t selected_pid = message.pid;

                //coda messsaggi type=pid
                message.msg_type = selected_pid;
                message.pid = selected_pid;

                if (msgsnd(msg_id_1, &message, sizeof(message.pid), 0) == -1) {
                    perror("Errore nell'invio del PID all'Atomo per la scissione");
                }
                else {

                //aggiorno memoria condivisa con le nuove operazioni
```

Successivamente attende STEP_ATTIVATORE_NSEC nanosecondi prima di rieseguire queste operazioni.

PROCESSO ALIMENTATORE:

Il processo alimentatore una volta avviato aggiunge 3 processi atomo ogni STEP_ATTIVATORE nanosecondi mediante le operazioni:

```
for (int i = 0; i < N_NUOVI_ATOMI; ++i)
        {
            pid_t pid = fork();
            if (pid == -1) {
                perror("Errore nella fork");
                kill(masterPid, SIGUSR1);
            } else if (pid == 0) { // Processo figlio (atomo)
                unsigned int seed = time(NULL) ^ getpid(); // Genera un seme unico
per ogni processo figlio
                char *args[] = {"./bin/atomo", generate_atomic_number(&seed),
alimPid_str, NULL};
                printf("ALIMENTATORE AGGIUNTO NUOVO ATOMOOO");
                execve(args[0], args, NULL);
                perror("Errore nell'esecuzione di execl");
                kill(masterPid, SIGUSR1);
            }
        }
```

Il seme del processo figlio viene utilizzato dalla funzione generate_atomic_number per generare il numero atomico casuale del processo atomo.