Relazione di Progetto Sistemi Operativi 2023/2024

Studenti:
Fiocco Nicolas , Corso A Matricola: 1052526 nicolas.fiocco@edu.unito.it
Floridia Giorgio Francesco , Corso A Matricola: 1054584 giorgio.floridia@edu.unito.it
Floriddia Tiziano Jhonny , Corso A Matricola: 1041726 tiziano.floriddia@edu.unito.it

Il progetto è strutturato in un processo principale, tre processi secondari, due file di intestazione e un makefile.

Programmi all'interno della directory:

1. Master.c

   Il processo Master avvia la simulazione creando e gestendo una memoria condivisa, utilizzata per le variabili statistiche, insieme ai semafori necessari per garantire la mutua esclusione. Queste variabili, che i processi possono modificare, vengono infine lette e visualizzate dal Master. Alcune di esse agiscono come segnali di errore per i processi, indicando quando è necessario interromperli. Il Master genera vari processi figli utilizzando la funzione fork, seguita da execve() per eseguire il codice dei processi figli. Inoltre, è stato implementato un secondo metodo di comunicazione inter-processo (IPC) tramite pipe. In questo caso, il Master crea una pipe per ciascun processo figlio e passa il file descriptor di lettura e l'indirizzo della memoria condivisa. Il Master stampa ogni secondo le statistiche, in più controlla le variabili di errore per determinare se la simulazione debba terminare oppure no. Al termine, il Master esegue la funzione di terminazione(), che disattiva il timer, aspetta che i processi figli si stacchino dalla memoria condivisa e libera le risorse prima di concludere l'esecuzione.

2. Atomo.c

   Gli atomi, dopo l'inizializzazione, controllano la presenza di errori. Se leggono una variabile condivisa che indica che il processo attivatore richiede la scissione, verificano se il loro numero atomico supera il valore minimo, se quindi il numero atomico non è sufficientemente grande l'atomo viene classificato come scoria e termina il processo. Se ciò non accade, l'atomo si scinde tramite la funzione fork() e crea un nuovo processo atomo con execve(), sprigionando energia. La funzione di terminazione() è stata implementata per gestire correttamente la conclusione del processo.

3. Attivatore.c

   Questo processo segnala agli atomi di scindersi modificando il valore di una variabile nella memoria condivisa, chiamata "attivazioni", che funge da contatore

incrementale. Gli atomi confrontano il loro contatore con quello della memoria condivisa; se coincidono, procedono con la scissione.

4. Alimentatore.c
   Questo processo ha il compito di generare nuovi atomi utilizzando la funzione fork() e execve(). Gli errori e i fallimenti vengono gestiti tramite le necessarie procedure di terminazione.

5. Makefile
   Per facilitare la compilazione, abbiamo creato un file "Makefile". Abbiamo scelto gcc come compilatore e impostato le opzioni di compilazione richieste (-Wvla, -Wextra, -Werror). Il makefile include i comandi per la corretta compilazione dei programmi e la possibilità di eseguire "make clean" per rimuovere gli eseguibili e i file oggetto generati.

6. Struttura.h
   Qui vengono definite le variabili e i semafori da allocare nella memoria condivisa.

7. Costanti.h
   In questo file sono dichiarate le costanti utilizzate durante la simulazione nei vari processi.