Progetto 2022

La Consegna prevedeva di sviluppare un programma che simulasse lo scambio di transazioni tra vari utenti mediante l'utilizzo di nodi.

Ho deciso di strutturare il programma partendo dallo sviluppo del processo che avrebbe gestito tutta la simulazione chiamato *MASTER.C*.
Per prima cosa ho implementato una funzione che caricasse i dati necessari allo svolgimento da file così che, come da indicazioni, i dati che sarebbero dovuti essere letti a tempo di esecuzione, potessero essere modificati senza compilazione. Dopodiché, avendo così il numero dei processi utente e nodo, master è in grado di generare i suddetti processi tramite una funzione dedicata per nodi e utenti, dove viene invocata le funzioni *Fork()* ed *execve()*.
Inizialmente avevo ipotizzato di passare come argomenti della *execve()* alcuni dati necessari allo svolgimento dei processi figli ma conseguentemente a vari test e allo sviluppo del codice è risultato maggiormente adeguato far caricare, con la stessa funzione utilizzata da master, i dati contenuti nel file *master_data.txt.*
Essendo necessario memorizzare i *pid* di tutti i processi figli, ho trovato in due *shared memory SysV* dedicate, la soluzione, che li rendesse disponibili alla condivisione tra tutti i processi.
Oltre alle shared memory ho trovato necessario implementare una *message queue SysV,* che permettesse la condivisione delle transazioni e *due semafori Posix* che consentissero la sincronizzazione dell'accesso in zone di codice e strutture dati sensibili.
La scelta di utilizzare due semafori al posto di uno emerge dalla volontà di rendere maggiormente fluida la simulazione.
L'ultima struttura dati condivisa, shared memory SysV, è dedicata al libro mastro, struttura al cui interno i processi nodo scriveranno blocchi di dimensione prestabilita mentre i processi utente e il master leggeranno le transazioni precedentemente eseguite.
Il corpo del processo master consiste in un ciclo do-while che stampa ogni secondo lo stato dei processi figli aggiornato e termina quando la durata della simulazione ha raggiunto il tempo stabilito, è avvenuta la terminazione di tutti gli utenti, oppure il libro mastro ha raggiunto lo spazio massimo disponibile. Concluso il ciclo, master, provvederà a mandare *SIGINT* a tutti i processi user ancora attivi e ai nodi, aspetterà la terminazione di ogni e stamperà tutti i dati richiesti. In conclusione questo processo gestirà la corretta terminazione di tutte le strutture *IPC*.

USERS.C è il programma che gestisce l'esecuzione dei singoli utenti. Dopo il controllo degli argomenti passati dal *execve(),* quali secondi e nanosecondi dell'inizio della simulazione da parte di master, e la dichiarazione della *sigaction()* che gestirà i segnali ricevuti *(SIGINT, SIGUSR1, SIGUSR2)*, procede ad allocare una struttura dedicata a conservare le transazioni calcolate e inviate ma non ancora presenti in master_book.
Il corpo di USERS.C è un ciclo while nel quale viene eseguito il calcolo di una transazione random ogni lasso di tempo calcolato in modo randomico compreso negli estremi indicati.
Transaction_data() è la funzione che effettivamente esegue la transazione calcolando il budget facendo la somma aritmetica tra tutte le transazioni eseguite e ricevute presenti in *master.book* e *buffer_pre_book*; la funzione restituisce 1 nel caso in cui non riesca a eseguire una transazione, a casa di varie possibili situazioni come budget terminato o terminazione user, così da aumentare il conto di *try* di 1 e 0 altrimenti, azzerando il conto dei *try*.

USER.C gestisce in SIGUSR1 il caso in cui la transazione sia stata inviata a un nodo che non poteva gestirla mentre SIGUSR2 esegue una transazione a comando dell'utente, gestendo *le interrupted system call*.

NODES.C controlla anch'egli gli stessi argomenti di USERS.C e alloca la *transaction_pool*, struttura che memorizzerà le transazioni a blocchi per poi scriverle nel libro mastro.
Il corpo di NODES.C conta in un ciclo while in cui vengono lette transazioni fino al limite del blocco meno 1, limite che una volta raggiunto farà si che nodes aggiunga una transazione personale con tutti i rewards gestiti in quelle transazioni. Una volta raggiunto il limite della transaction_pool uscirà dal ciclo per entrare in uno infinito in cui, quando riceverà una transazione manderà SIGURS1 all'user sender.
La terminazione di nodes avviene solamente tramite SIGINT emesso da MASTER.C.

Per rispettare il requisito della divisione in moduli del codice, e per comodità, ho implementato una libreria my_lab.h nella quale ho inserito tutte le funzioni necessarie a tutti i processi e tutte le variabili e strutture condivise.