RELAZIONE PROGETTO DI SISTEMI OPERATIVI

ANNO 2021/2022

Realizzato da:

- *Cognome: PEDDONI*
- *Nome: STEFANO*
- *Matricola: 839138*
- *Corso: B*

- *Cognome: LAGUARDIA*
- *Nome: DAVIDE*
- *Matricola: 822358*
- *Corso: B*

- *Cognome: CRISTIANO*
- *Nome: GIULIA*
- *Matricola: 824010*
- *Corso: A*

STRUTTURA DEL PROGETTO

FILES	DESCRIZIONE
master.c master.h	Parte principale del progetto, dove vengono creati gli utenti e i nodi, con varie funzioni che servono alla lettura dei file e stampa delle varie statistiche finali. Il file master.c è collegato ad un header file contenente le dichiarazioni delle funzioni.
user.c user.h	Gestione degli utenti con invio delle transazioni mediante l'uso di una coda di messaggi, ai nodi con rispettive funzioni random. Il file user.c è collegato ad un header file contenente le dichiarazioni delle funzioni.
node.c node.h	Ricezione delle transazioni da parte degli utenti con la coda di messaggi e memorizzazione all'interno di una transaction pool. Il file node.c è collegato ad un header file contenente le dichiarazioni delle funzioni.
sem.c sem.h	Qui vengono gestite le funzioni inerenti ai semafori. Il file sem.c è collegato ad un header file contenente le dichiarazioni delle funzioni.
shared_variables.h	All'interno di questo header file troviamo le variabili e le strutture che vengono utilizzate all'interno di ciascun files (user,node) e alcune define.
makefile	Contiene le direttive per compilare il progetto: con make vengono creati i file bin e build e con ./master viene fatto partire il progetto. Per una pulizia si utilizza make clean.
config	All'interno della cartella config abbiamo tre file di testo diversi con i valori di ogni parametro che troviamo all'interno della struttura "configuration" in shared_variables.
esegui.sh	Script di avvio del progetto che si occupa di eseguire il make, pulire i processi zombie, IPC non rimosse e avviarlo. N.B. nel caso in cui, una volta eseguito "./esegui.sh" dovesse apparire il messaggio : "comando non trovato", bisogna digitare sul terminale queste due stringhe: "chmod u+r+x esegui.sh" , "sed -i -e 's/\r$//' esegui.sh".

SCELTE IMPLEMENTATIVE

All'interno del MASTER, file principale che fa partire il progetto e si occupa della corretta esecuzione della simulazione, abbiamo realizzato:

- Funzioni come lo spawn degli utenti e lo spawn dei nodi (dove viene eseguita la fork e passati i vari argomenti presenti tramite execve).
- Creazione di memorie condivise come:
    - shm_mastro -> memoria condivisa del libro mastro.
    - shm_user -> memoria condivisa dell'utente, contente anche una struttura inserita all'interno di shared_variables dove possiamo trovare il pid di ogni singolo utente, il budget e il numero di utenti che terminano premutaramente (memorizzato come un array che parte dal numero effettivo di utenti, es: 100 (config1), e viene decrementato all'interno del file user quando i tentativi raggiungono il numero di so_retry).
    - shm_node -> memoria condivisa del nodo, contenente anch'essa una struttura inserita all'interno di shared_variables dove possiamo trovare il pid di ogni singolo nodo, una variabile che serve a memorizzare la coda di messaggi all'interno del nodo stesso e un'ultima per memorizzare le transazioni rimanenti nella transaction pool.
    - shm_trans_discard -> memoria condivisa, contenente le transazioni scartate che non vengono memorizzate all'interno della transaction pool.
    - shm_discard -> memoria condivisa, contenente la dimensione delle transazioni scartate, il puntatore viene realizzato come un array che parte da 0, poiché all'inizio non ci sono transazioni scartate.
- Creazione di semafori.
- Id e parametri passati come argomento sia all'interno dello spawn degli utenti che dei nodi con i valori a run time presenti nei vari file di configurazione.
- Ricezione dei segnali:
    - SIGINT -> avviene attraverso il comando CTRL+C, il programma termina in qualsiasi momento pulendo la memoria condivisa, i semafori e la coda di messaggi attraverso la clean_up.
    - SIGALRM -> avviene quando il tempo (in secondi) finisce, appare un messaggio e il programma procede con le statistiche finali.
    - SIGUSR1 -> avviene quando lo spazio sul libro mastro (REGISTRY_SIZE) si riempie prima della terminazione.
    - SIGUSR2 -> avviene quando gli utenti terminano tutti e non ne rimane più nessuno in vita.

    Sia in SIGALRM, SIGUSR1 che in SIGUSR2 una volta ricevuto il messaggio che notifica quello che è avvenuto, gli ulteriori utenti e nodi vengono terminati con SIGTERM all'interno di master e vengono stampate le statistiche finali per poi pulire il tutto.

All'interno di USER, viene gestito l'invio delle transazioni da parte di ogni singolo utente ai processi nodo, viene anche gestita la funzione per le transazioni scartate per far si che un utente prima dell'invio del messaggio al nodo verifica se la tripletta (sender, receiver e timestamp) non sia già presente all'interno.

- Se il budget è maggiore o uguale a 2 vengono scelti random diversi fattori: utente, quantità, reward e node, per poi essere inseriti all'interno del buffer.transaction utile poi per l'invio del messaggio al nodo.
- Se il budget è minore di 2 il processo non invia alcuna transazione.

All’interno di NODE, viene gestita la ricezione attraverso la msgrcv delle transazioni inviate dal singolo utente, se va a buon fine viene inserita all’interno della transaction pool, altrimenti viene scartata all’interno della memora condivisa shm_trans_discard.

- All’interno delle transazioni viene aggiunta all’ultimo blocco una transazione di reward contenente:
    - Timestamp (valore attuale di clock_gettime).
    - Sender: -1 definita attraverso una MACRO all’interno di shared_variables.
    - Receiver: l’identificatore del nodo corrente.
    - Quantità: la somma di tutti i reward delle transazioni incluse nel blocco.
    - Reward: 0

Una volta scritte con successo le transazioni all’interno del blocco, vengono eliminate dalla transaction pool.

STATISTICHE FINALI

Durante l'esecuzione del progetto, ogni secondo vengono stampati gli utenti e i nodi con maggior e minor budget (essendo tanti gli utenti e i nodi abbiamo inserito all'interno di shared_variables due define per far capire alla stampa iniziale che se gli utenti e i nodi sono troppi, di stampare soltanto quelli più significativi); insieme al numero di utenti e nodi effettivi per ogni secondo (dato che gli utenti possono terminare premutaramente).

In questo caso possono accadere tre cose:

- Se il messaggio che appare è: sono passati "tot" secondi allora il programma ha terminato il numero di secondi scelto dall'utente.
- Se il messaggio che appare è: users insufficienti, allora il numero di utenti è stato decrementato fino ad arrivare a 0.
- Se il messaggio che appare è: finito spazio su libro mastro, allora lo spazio che contiene il registro si riempie prima della fine del tempo.
- Infine, abbiamo gestito un segnale con SIGINT, che se digitato su tastiera CTRL+C, il programma termina e fa anche una pulizia.

Dopo la ricezione del segnale da parte del master, vengono visualizzate a terminale:

- Numero di utenti che terminano prematuramente.
- Numero di blocchi scritti.
- Stampa dei blocchi se necessario.
- Bilancio finale di utenti e nodi.
- Numero di transazioni ancora presenti nella pool di ogni singolo nodo.

COMPILAZIONE

La compilazione del progetto è gestita tramite l'utilizzo dell'utility make. Esiste infatti un makefile che serve a far compilare tutti i file per la simulazione.

Per far compilare il tutto basta digitare nel terminale "make" e verrà compilato il codice sorgente prendendo in considerazione le varie librerie necessarie per la compilazione.

Sono state inseriti questi requisiti richiesti nel testo:

- essere realizzato sfruttando le tecniche di divisione in moduli del codice,
- essere compilato mediante l'utilizzo dell'utility make
- massimizzare il grado di concorrenza fra processi
- deallocare le risorse IPC che sono state allocate dai processi al termine del gioco
- essere compilato con almeno le seguenti opzioni di compilazione:
  gcc -std=c89 -pedantic
- poter eseguire correttamente su una macchina (virtuale o fisica) che presenta parallelismo (due o piu processori).

In più è stato inserito -D_DEBUG che se assume il valore 1 vengono stampate delle statistiche che sono state inserite da noi durante l'esecuzione del progetto (utile per capire se ci sono stati problemi nel corso della realizzazione) e se assume il valore 0 ogni printf che ha nell'if (_DEBUG) non viene stampato nulla.

Al fine di velocizzare la fase di test, è stato creato uno script bash chiamato *esegui.sh* che contiene:

- Rimozione di eventuali processi zombie.
- Deallocazione e rimozione di eventuali risorse IPC.
- Compilazione del codice.
- Avvio del progetto.