RELAZIONE DI SINTESI SULLE SCELTE PROGETTUALI
A.A. 2021/2022

All'interno della cartella è presente un file *macro.txt* che contiene tutti i parametri da leggere a tempo di esecuzione. Vengono letti dalla funzione *read_macro(argv[1])* presente nei file *master.c*, *users.c* e *nodes.c.* Nel file *header.h* sono incluse tutte le librerie che servono per la corretta esecuzione del programma, sono definite le chiavi per semafori e memorie condivise, sono definiti i parametri da leggere a tempo di compilazione e sono definite le seguenti strutture:

- **transaction**: struttura di una transazione che contiene i campi timestamp, sender, receiver, quantity e reward.
- **p_info**: struttura che permette di avere informazioni su un processo e contiene i campi proc_pid (PID del processo), proc_balance (bilancio del processo), pos (posizione del processo nella memoria condivisa), term (processo terminato o meno)
- **block**: struttura di un blocco del libro mastro che contiene i campi array_trans (array di transazioni di dimensione SO_BLOCK_SIZE) e id (identificativo numerico del blocco)
- **message**: struttura del messaggio da inviare alle code di messaggi che contiene i campi mtype (tipo del messaggio) e mtext (testo del messaggio)

Nei file *master.h*, *users.h* e *nodes.h* sono dichiarate le firme dei metodi, utilizzati nei rispettivi file.
Nel file *master_functions.c* sono implementati i metodi che vengono usati in *master.c*, nel file *users_functions.c* sono implementati i metodi che vengono usati in *users.c* e nel file *nodes_functions.c* sono implementati i metodi che vengono usati in *nodes.c*.

MASTER

Inizialmente vengono letti i parametri dal file *macro.txt* e vengono assegnati i valori alle macro. Usa la system call *sigaction()* per gestire i segnali e viene definita una funzione *handle_signal(sig)* per gestire i segnali che possono occorrere durante la simulazione. Crea la memoria condivisa per il contatore dei blocchi nel libro mastro, per la struttura *p_info* con le informazioni sui processi e per il libro mastro. Crea il semaforo per scrivere nel libro mastro e crea il semaforo per aspettare la creazione di tutti i processi. Successivamente, attraverso un ciclo di lunghezza SO_USERS_NUM + SO_NODES_NUM e una *fork()*, crea i processi figli users e nodes, li inserisce nella memoria condivisa e richiama *execvp()* sia per gli utenti che per i nodi. Finita l'attesa di creazione dei processi, grazie al semaforo del set SEMKEY_P_INFO, decrementa quest'ultimo e parte la simulazione. Setta un *alarm()* di SO_SIM_SEC ed entra nel ciclo *while(1)*, dove viene impostato un secondo di attesa, per le stampe successive, con la

funzione *one_sec_waited_master().* Aggiorna il numero di utenti terminati, li sottrae a quelli attivi e controlla se sono terminati tutti o se ne è rimasto in vita solo uno (che non può mandare transazioni a nessun altro). (In questi due casi stampa la terminazione con la funzione *print_end()* e invia un segnale di fine simulazione) Con la funzione *print_proc()* stampa il PID dei processi, il loro bilancio e se sono terminati o meno. Infone, stampa il numero di utenti terminati, gli utenti e i nodi attivi e i blocchi scritti nel libro mastro. Finita la simulazione dealloca tutte le risorse IPC.

USERS

Inizialmente vengono letti i parametri dal file *macro.txt* e vengono assegnati i valori alle macro. Usa la system call *sigaction()* per gestire i segnali e viene definita una funzione *handle_signal(sig)* per gestire i segnali che possono occorrere durante la simulazione. Salva nelle variabili gli identificatori di memoria per il contatore di blocchi nel libro mastro, per la struttura *p_info* con le informazioni sui processi e per il libro mastro. Salva anche l'identificatore del set di semafori SEMKEY_P_INFO. Attende il via libera per partire con la simulazione attraverso i semafori, utilizzando la funzione *wait_for_zero()*. Con un ciclo *for(j = 0; j < SO_USERS_NUM; j++)* recupera la sua posizione in memoria condivisa e inizializza tre variabili: *balance*, *index_blocks* e *attempt*, rispettivamente il bilancio del processo, l'indice di blocchi letti e i tentativi di invio transazione. Setta un *alarm(rand() % SO_SIM_SEC + 1)* per generare il segnale SIGALRM scelto da noi per inviare una transazione. Entra nel ciclo *while(attempt != SO_RETRY)* da cui esce quando non è riuscito ad inviare una transazione per SO_RETRY volte consecutive. Dentro al *while(attempt != SO_RETRY)* calcola il bilancio con *budget_calc()*, inizializza la transazione con *trans_init()* e la imposta e la invia al nodo tramite *trans_send().* Uscito dal *while(attempt != SO_RETRY)* aggiorna il campo *term* a *true* della *struct p_info* e termina.

NODES

Inizialmente vengono letti i parametri dal file *macro.txt* e vengono assegnati i valori alle macro. Usa la system call *sigaction()* per gestire i segnali e viene definita una funzione *handle_signal(sig)* per gestire i segnali che possono occorrere durante la simulazione. Crea la coda di messaggi, la "inizializza" con IPC_STAT, setta la dimensione di byte massimi nella coda con *buf.msg_qbytes = SO_TP_SIZE * sizeof(struct transaction)* e aggiorno la struttura *msqid_ds* con la nuova dimensione massima. La transaction pool, quindi, viene creata dal nodo con l'ausilio della coda di messaggi, che avrà massimo SO_TP_SIZE transazioni in coda. Salva nelle variabili gli identificatori di memoria per il contatore di blocchi nel libro mastro e per il libro mastro. Salva nelle variabili anche l'identificatore del set di semafori SEMKEY_P_INFO e l'identificatore del set di semafori SEMKEY_LEDGER. Attende il via libera a partire con la simulazione attraverso i semafori, con la funzione *wait_for_zero()*. Entra nel ciclo *while(1)* dove, con IPC_STAT, aggiorna *buf* per vedere quanti messaggi ci sono in

coda e, se ce ne sono abbastanza, crea il blocco leggendo SO_BLOCK_SIZE – 1 transazioni dalla transaction pool con la *msgrcv().* Nell'ultima posizione del blocco aggiunge la transazione di reward con la funzione *add_reward_trans().* Fa la *reserve_sem()* per permettere la scrittura del blocco sul libro mastro. Controlla se c'è spazio nel libro mastro, aggiorna l'id del blocco con il numero di blocchi correnti, simula l'attesa di elaborazione con la funzione *one_sec_waited_nodes()*, scrive il blocco nel libro mastro e incrementa il contatore di blocchi del libro mastro. Dopo la scrittura del blocco controlla se lo spazio nel libro mastro è esaurito e se ciò accade invia un segnale SIGUSR1 al master e mette in pausa il processo in attesa di SIGTERM, che, quando arriva, dealloca le risorse IPC e termina. Alla fine del *while(1)* fa la *release_sem().*