simulazione_transazioni

Programma sviluppato per il progetto di Sistemi Operativi del corso di laurea in informatica, presso l'Università degli studi di Torino anno 2021/2022.

Compilazione e uso

Per compilare il programma è necessario avere installato sulla propria macchina linux `gcc` e `make`.

Nella cartella configs ci sono degli script per avviare il programma: basta scaricare il progetto, fare il comando `make` nella cartella principale del progetto (verrà creato l'eseguibile `master`), e infine avviare uno degli script dalla cartella in cui è presente l'eseguibile (per esempio, `./configs/conf1.sh`)

Funzionamento

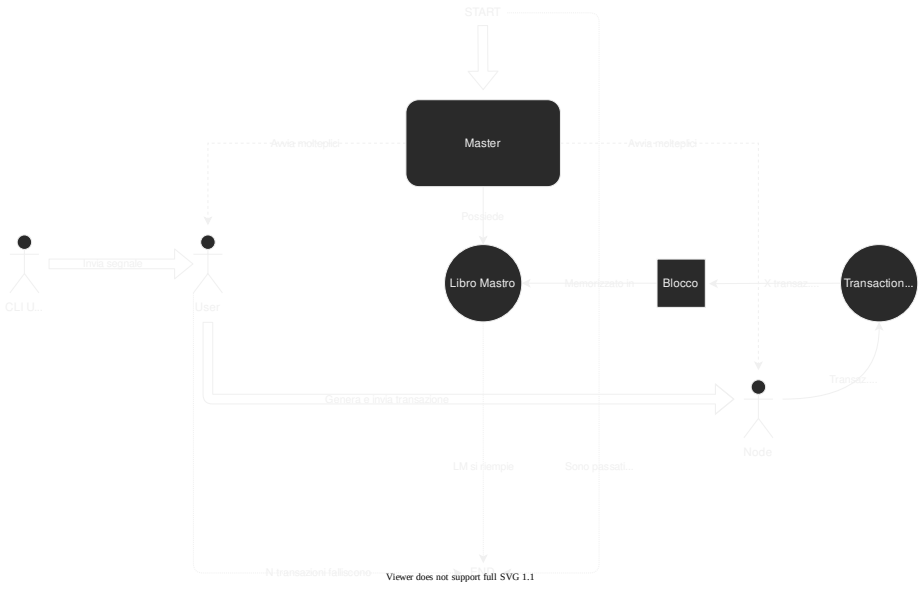

Figure 1: Diagramma di funzionamento programma

Master

Processo avviato all'avvio del programma, incaricato di:

1) Ottenere le shared memory che ci servono e i semafori necessari
2) Avvio dei processi Node e User, con relativo popolamento degli array nelle shared memory contenenti informazioni quali bilancio, stato del processo, PID, amici, ID message queue...

3) Invio del segnale `SIGUSR1`, segnale che sancisce l'inizio della simulazione, inviato dopo che tutti i processi sono stati avviati (e le relative informazioni inserite nell'array condiviso)
4) Avvio del processo incaricato di stampare ogni secondo lo stato della simulazione
5) Avvio del processo incaricato di avviare ulteriori processi Node

Infine, dopo che la simulazione è terminata (può terminare per: mancanza di spazio nel libro mastro, mancanza di bilancio dei processi User, timeout definito con `SO_SIM_SEC`), verrà stampato un resoconto finale dello stato dei processi Node e User e la ragione di terminazione.

User

Processo secondario che viene avviato dal processo Master, si occupa d'inviare le transazioni ai nodi tramite coda di messaggi. Il suo bilancio viene inizializzato a partire da un determinato budget, di cui, a ogni transazione, ne verrà inviata una porzione uguale a un valore casuale tra 2 e il bilancio attuale.

Molte delle operazioni che esegue il processo User prevedono l'utilizzo della memoria condivisa, in particolare viene spesso utilizzata `[User](#user)s_pid`: questo array contiene le informazioni di tutti i processi utente creati dal Master. Accedendo all'array utilizzando l'indice fornito dal Master al momento dell'avvio (`[User](#user)_position`), è possibile prelevare le informazioni pertinenti al processo User d'interesse: il suo bilancio, il suo PID e il suo stato.

Il processo User può terminare in 2 modi differenti:

1) Quando fallisce un determinato numero di transazioni: una transazione fallisce quando lo User ha un bilancio inferiore a 2, e se fallisce `SO_RETRY` transazioni di fila il processo termina.
2) Quando scade il tempo: la simulazione ha una durata prestabilita da configurazione (`SO_SIM_SEC`), e a scadere del tempo il processo termina.
3) Quando il libro mastro è pieno.

La terminazione degli utenti viene gestita tramite un flag `stop`, alzato da parte del processo User o dal processo Master quando si soddisfa una delle condizioni descritte sopra

Node

Processo secondario che gestisce le transazioni, che ricevono da parte dei processi User tramite coda di messaggi. Memorizza le transazioni ricevute in una transaction pool di grandezza prestabilita (`SO_TP_SIZE`), e processa le transazioni in blocchi anch'essi di grandezza prestabilita (`SO_BLOCK_SIZE`).

Anche esso fa ampio uso di risorse presenti in memoria condivisa, principalmente l'array `[Node](#node)s_pid` e il libro mastro `ledger`. Similmente al processo User, Node usufruisce di `[Node](#node)s_pid` per prelevare informazioni sul

nodo d'interesse anch'esso tramite un indice `[Node](#node)_position` fornito dal Master all'avvio. Tramite esso, è possibile conoscere il bilancio del nodo (calcolato sul reward di ogni transazione), il suo PID, il suo stato, l'ID della sua message queue e numero di transazioni presenti nella sua transaction pool.

Il Node, diversamente dai processi User, termina soltanto quando il flag `stop` viene sollevato, ovvero quando tutti gli utenti sono terminati.