**Relazione Dadone Luca – Versione da 30**

* Ho realizzato processi user e node attraverso due cicli di fork() ed execve() dal master.
* Ho gestito la fase di sincronizzazione all’avvio attraverso 2 semafori per massimizzare il grado di concorrenza:
  + Semaforo "pronti tutti" impostato alla cardinalità dei figli  (SO\_USERS\_NUM + SO\_NODES\_NUM) .

Il master si mette in attesa dello 0 (che ogni figlio sia stato avviato).

Ogni processo decrementa con una reserveSem tale semaforo all'avvio.

* + Successivamente, per massimizzare la concorrenza tra utenti, ogni user rimarrà in attesa dello 0 di un secondo semaforo: "liberi tutti".

Tale semaforo è inizializzato in origine a 1 dal master, così che, una volta che il master riprende ad eseguire perché i figli son tutti pronti, esegue una reserveSem su tale semaforo e gli utenti potranno cominciare ad eseguire.

* Ho reso possibile l'inserimento dei parametri di configurazione a tempo di esecuzione tramite più modalità:
  + 0 default : utilizza i valori di default
  + 1,2,3 : utilizzano valori delle tre configurazioni fornite nella consegna (per rendere più veloce la fase di test)
  + 4 **manuale** attraverso stdin: permette l'inserimento dei valori da testiera all'utente.

Ottenuti tali parametri li ho memorizzati in una struttura conf, salvata in shm in modo da renderla accessibile da utenti e nodi.

* Per la condivisione dei pid tra processi ho utilizzato:
  + Un elenco degli user (pid) implementato attraverso un array dal master con una calloc() e condividendo tale array con gli user tramite shm.
  + Stessa scelta per comunicare i nodi agli user.
  + Per comunicare i pid dei nodi friends ai vari processi nodo ho invece utilizzato una message queue, salvando i nodi in una **lista dinamica** (privatamente per ogni processo) visto che il numero complessivo di nodi può variare durante l'esecuzione della simulazione.
  + Lista dinamica anche usata nel master ma arricchita da altre informazioni quali il numero di transazioni presenti a fine simulazione nella tranasction\_pool di ogni nodo, utile in fase finale di stampa.
* Ho gestito l'invio/ricezione di transizioni da utente a nodo tramite un'ulteriore apposita msg queue. Transizioni salvate privatamente da ogni nodo attraverso una lista dinamica transaction\_pool.
* La condivisione del libro mastro tramite shm, e per l'accesso a tale libro ho deciso di implementare il meccanismo di **Lettori-Scrittori** così che più utenti (+ il master) possano leggere contemporaneamente dal libro mastro ma un solo nodo alla volta possa scriverci bloccando possibili lettori. Attraverso un semaforo mutex per l'accesso alla variabile numLettori in memoria condivisa e un semaforo scrivi per bloccare possibili lettori in fase di scrittura di un nodo (stesso procedimento visto a lezione e a teoria)

(inizioLettura() fineLettura() ..).

* + Per la stampa in master() ho utilizzato **nanosleep**() per l'attesa inattiva di 1sec implementando un meccanismo che mi permetta di, in seguito all’interruzione per la ricezione di un segnale come ad esempio sigchld, riprendere la pausa per il numero di nanosecondi rimanenti della pausa interrotta.

Nanosleep() utilizzata anche per le altre pause inattive descritte dalla consegna.

Una sola funzione di stampa mi permette di gestire le stampe periodiche del master, e quella di riepilogo con i dati richiesti dalla consegna (ragione di arresto ecc)

Ho implementato la stampa solo dei dati più significativi se vi sono troppi user per essere visualizzati a terminale, anche nella stampa di riepilogo, dove invece i nodi(solitamente in numero minore) vengono stampati tutti per poter meglio osservare il funzionamento dell’applicazione relativo alla versione da 30.

* + Ho gestito ed utilizzato vari segnali come dal master:
    - SIGCHLD per salvarmi il numero di **utenti attivi** (decrementando a ricezione avvenuta)
    - SIGALARM per portare l’arresto dell’esecuzione dopo SO\_SIM\_SEC secondi
    - SIGURS1 per notificare il master da parte di un nodo che ha raggiunto la **capienza massima del libro mastro** e deve quindi terminare la simulazione
    - SIGURS2 inviato da un nodo al master quando **una transazione ha compiuto SO\_HOPS\_NUM salti** e quindi viene inviata al master il quale **crea un nuovo nodo**, gli invia la transazione (message queue), gli invia SO\_FRIENDS\_NUM nodi amici che ho deciso di scegliere casualmente (message queue), e invia tale nodo a SO\_FRIENDS\_NUM nodi casuali **notificando tali nodi con un segnale** **SIGURS2.**
    - SIGURS1 inviato periodicamente dal master ad uno user **scelto a caso** per testare **l’invio di una transizione inseguito alla ricezione di un segnale**, (se il bilancio lo permette)
    - SIGTERM inviato dal master ai figli a termine simulazione, gestito dai nodi che comunicano al master la propria tp size e terminano.
  + Ho implementato un’ulteriore coda di messaggi per ricevere nel master, dai nodi, il numero di **transizioni presenti nella propria pool al momento della loro terminazione** in fase di arresto- riepilogo.
  + Inoltre ogni nodo invia periodicamente una transizione ad un suo friend attraverso la stessa coda di massaggi utilizzata dagli user, una volta che ha raggiunto SO\_TP\_SIZE decrementando il campo **hops**.  Tale campo viene invece risettato al valore di default quando raggiunge il valore zero e la transizione viene mandata al master (**notificato da un segnale** SIGURS2), il quale crea un nuovo nodo per processarla ecc.
  + Ho infine utilizzato **maschere di messaggi** per bloccare la ricezione di alcuni segnali durante l'esecuzione di operazioni che non voglio vengano interrotte come l'accesso al libro mastro.