Relazione Progetto di Sistemi Operativi 2021-22 --- Norbiato Jacopo, 954965

Schema generale :

Ogni processo protagonista (master, user, node) nell'esecuzione è contenuto nel corrispettivo file.c a cui è associato un proprio header.h e un altro file_fun.c contenente la specificazione delle funzioni e le variabili esterne.
Ogni processo fa riferimento al file header.h generale ove sono specificate le librerie adoperate e inizializzate le struct necessarie al corretto svolgimento.
Il processo master.c è deputato dell'avvio dell'esecuzione e alla genesi dei processi user e node attraverso due fork ( ) successive; esso, inoltre è responsabile della pubblicazione costante di un monitoraggio delle statistiche durante il run-time, nonché del riepilogo a terminazione processo.

Preparativi :

Master.c legge i parametri da utilizzare nella simulazione dal file "opt.conf" grazie alla funzione configura_macro( ), quindi procede ad ottenere le strutture IPC dedicate:

- 3 memorie condivise così suddivise :
    a) memoria_block = conta le pagine del libro mastro trascritte durante il run-time
    2) mem_stamp = memoria utilizzata per ottenere informazioni sullo stato dei processi user e node
    3) mem_lib = memoria effettiva del libro mastro ove sono contenute tutte le transazioni

- 1 struttura semafori, costituita da 4 semafori così ripartiti :
a) sem(0) : regola i tempi del processo master.c
b) sem(1) : regola i tempi dei processi user
c) sem(2) : regola i tempi dei processi node

Simulazione :

Master.c
Procede alla genesi dei processi user e node con due successive fork ( ), gestendo i loro tempi tramite l'impostazione del valore dei semafori 1 e 2
Quindi rilascia i semafori all'avvio dell'esecuzione ed entra in un ciclo infinito for (;;) dove stampa ad intervalli regolari (attesa_master( )) lo stato di progresso dei processi user e mantiene il conto di quanti processi user abbiano terminato prematuramente la loro esecuzione
Quindi effettua la stampa finale print_Stats( ) qualora siano occorse le condizioni di terminazione.

User.c
Parimenti a master.c, provvede a leggere le macro riguardanti la sua esecuzione dal file esterno "opt.conf" e inserisce il proprio pid nella struct p_details tramite il corrispondente pointer *ptr.
Quindi, una volta che la sua esecuzione viene riavviata dal master, entra in un ciclo while ( ) il cui variante è il contatore di fallimenti "fails". L'accesso alla sezione critica è gestito dal semaforo 1, che viene riservato e, dopo le operazioni, rilasciato per lasciare la risorsa ad altri user e permettere il giusto parallelismo nell'esecuzione.
Ora user calcola il budget corrente scorrendo il libro mastro e cercando tutte le transazioni in cui è coinvolto (tramite corrispondenza del pid), quindi vi sono 2 casi:
a) user è sender, quindi sottraggo la quantità inviata ad altro user dal budget iniziale
b) user è receiver, quindi aggiungo la quantità ricevuta da altro user al budget iniziale

Dunque, se budget >= 2, provvede al calcolo della quantità da inviare ad altro user attraverso la funzione user_quantity( ), cerca un user candidato tramite user_random_pick( ), imposta il timestamp e seleziona casualmente un pid node a cui inviare la transazione sotto formato di struct mymsg.
Quindi termina simulando un tempo di attesa tramite la funzione waiting_trans_gen ( ) e ritorna nel ciclo while di partenza.

Node.c
Procede allo stesso modo di user per ottenere le macro utilizzate, quindi procede alla creazione di una coda messaggi dove recuperare le struct mysmg inviate da user e imposta la dimensione avvalendosi della funzionalità msg_qbytes della struct msqid_ds associata.
Quindi entra in un ciclo infinito e come user accede alla sezione critica riservando/rilasciando il rispettivo semaforo, sem(2); legge il numero di messaggi nella coda e se vi sono abbastanza messaggi da poter scrivere un blocco (dimensione SO_BLOCK_SIZE) del libro mastro, procede.
Quindi trascrive le transazioni nel blocco riservando a sé la transazione in ultima posizione che è definita dalla macro LAST_TX e contiene la somma di tutte le reward trascritte (valore che continua ad aggiornare nella struct p_details dedicata tramite il pointer *ptr).
Allorquando il numero di blocchi trascritti equivale alla piena capacità del Libro Mastro, node provvede ad inviare il segnale SIGUSR1 a master e si mette in pausa, attendendo che questi svolga i suoi compiti.
L’accesso alla trascrizione dei blocchi in libro mastro è protetto dal semaforo sem(3).
Quindi node rilascia la risorsa dell’accesso in sezione critica e ritorna nel ciclo infinito.

Terminazione :

Master provvede a comunicare la ragione della terminazione tramite la dinamica switch-case conservata nella handle_signal( ). Quindi effettua un riepilogo del Libro Mastro e dei processi, comunicando quanti blocchi siano trascritti sul totale e quante transazioni siano ancora pendenti nelle Transaction Pool (sotto forma di coda messaggi) di ciascun processo node.
Quindi termina rimuovendo tutti i i sistemi IPC.