Progetto Sistemi Operativi
Versione Completa

Lorenzo Lombardi, Emanuele Parusso, Federico Paschetta,

February 2022

1 Studenti

1.1 Lorenzo Lombardi, mat. 948049, lorenzo.lombardi@edu.unito.it

1.2 Emanuele Parusso, mat. 946784, emanuele.parusso@edu.unito.it

1.3 Federico Paschetta, mat. 948420, federico.paschetta@edu.unito.it

2 Presentazione progetto e compilazione

Il progetto è composto da tre file principali con estensione .c, "master.c", "ProcessoNodo.c" e "ProcessoUtente.c", che eseguono rispettivamente il codice del processo "padre", del processo nodo e del processo utente. Per facilitare la compilazione è stato aggiunto un makefile, in cui sono stati inseriti i comandi "make" per compilare e linkare ciascun file, rispettando le regole di "std-c89" e "pedantic" e "make clean" per eliminare eventuali file oggetto preesistenti e ricompilare. La simulazione viene iniziata con l'esecuzione del file "master". Per rendere più leggibile il codice, inoltre, è stato aggiunto un file header, "Utils.h", incluso in ciascun file .c, che contiene struct, variabili e metodi utili ai tre processi.

3 Inizializzazione ambiente

Il master inizialmente definisce tutte le variabili che serviranno per l'esecuzione, in particolare le chiavi per l'accesso agli oggetti IPC (una coda di messaggi per la comunicazione tra processi, due memorie condivise, una contenente il libromastro e una contenente PID e budget di nodi e utenti e lista di amici per ciascun nodo, e un'array di tre semafori per avvio simultaneo della simulazione e accesso in mutua esclusione per la scrittura del libromastro), le variabili che conterranno i parametri passati a runtime e le struct definite in Utils.h. In seguito vengono eseguite msgget, semget e due shmget, i cui valori di ritorno sono assegnati agli identificatori della coda di messaggi, dell'array di semafori e delle due memorie condivise, in modo da inizializzare gli oggetti necessari

all'esecuzione. Viene invocata la funzione "signal()", per indicare la gestione dei segnali, affidata al metodo handle-signal e salvato in una variabile d'appoggio il PID del master. A questo punto inizia la parte di simulazione visibile all'utente, con l'inserimento dei valori richiesti a runtime, passati al processo master con una scanf e l'avvio dell'alarm che rappresenta il countdown relativo al tempo di simulazione. Vengono fatti gli attach delle tre memorie condivise, poi attraverso le sprintf le chiavi per l'accesso agli oggetti IPC diventano stringhe, passabili ai processi al momento dell'execve, e vengono inizializzati due dei tre semafori nell'array in modo da non permettere a nessun processo di eseguire il proprio codice prima del "semaforo verde" da parte del master.

4 Creazione processi

Il master assegna all'array di argomenti da passare a ciascun processo i valori corrispondenti, inizializza l'array relativo ai budget a 0 e fa una switch, basata sul valore di fork, per creare i processi. A questo punto se il processo entra nel case -1, viene segnalato un errore, se entra nel case 0 (quindi il codice relativo al processo figlio) viene eseguita la rispettiva execve (con argomenti quelli stabiliti prima), se entra nel ramo default, ottiene come valore di return della fork il PID del processo figlio creato e lo inserisce nella corrispondente cella dell'array relativo ai PID nella memoria condivisa. Questa sequenza di azioni viene eseguita iterativamente per SO-NODES-NUM volte e in maniera identica per SO-USERS-NUM volte. Inoltre, soltanto per i processi nodo, una volta inizializzati, viene eseguito un ciclo for che permette a ciascun processo di selezionare i propri NUM-FRIENDS amici. Dopo questo il processo master fa la shmctl delle memorie condivise con IPC-RMID, in modo da eliminarle quando non avranno più processi "agganciati" e dà il via all'esecuzione dei processi nodo e utente, sbloccando i semafori relativi.

5 Comunicazione tra processi

5.1 Processo nodo

Il processo nodo prima definisce le variabili che serviranno durante l'esecuzione del codice e poi le inizializza facendo l'atoi degli argomenti che gli sono stati forniti dal processo master e, in seguito, fa lo stesso per tutte le strutture dati e gli oggetti IPC utili al processo nodo. Esso poi, attraverso la funzione clock-gettime, imposta il punto da cui partirà il timer a cui si farà riferimento nelle transazioni nel libromastro e ricerca l'indice che rappresenta la propria posizione nell'array di PID (e relativi budget) nella memoria condivisa. A questo punto il processo nodo, finché non viene terminato attraverso il segnale SIGTERM, fa un ciclo continuo in cui scrive nella transaction pool (array di transazioni in memoria locale) il contenuto della transazione ricevuto con la msgrcv. Se l'indice che rappresenta la quantità di transazioni in transaction pool, aggiornato ad ogni ricezione, è uguale (o maggiore) al valore SO-TP-SIZE, invece,

si aprono due opportunità: il processo nodo aumenta il valore della variabile hops, e invia la transazione, attraverso la msgsnd, ad un nodo scelto in maniera random tra gli amici del processo; se il valore di hops è uguale al valore di SO-HOPS, la msgsnd viene inviata al processo master. Il processo nodo, inoltre, si occupa di trascrivere, quando nella transaction pool è contenuto almeno un blocco di transazioni e non è ancora pieno il libromastro, il contenuto della transaction pool nel libromastro. Questo processo, dopo l'attesa (attraverso la nanosleep) di un tempo compreso tra i due valori dati a runtime, viene svolto in mutua esclusione, modificando i valori delle variabili della struttura del semaforo e invocando la funzione semop. Il nodo controlla che il libromastro non sia pieno (confrontando l'indice posizione, aggiornato ad ogni scrittura, con SO-REGISTRY-SIZE) e, con un ciclo che itera fino alla penultima cella della riga dell'array, scrive la riga di transazioni con i valori delle corrispondenti transazioni nella transaction pool. A questo punto ottiene il valore corrente del timer, riempie le variabili della transazione di reward, aggiorna il budget del nodo nella cella corrispondente nella memoria condivisa che contiene le informazioni di tutti i processi e incrementa l'indice relativo all'ultima posizione scritta nel libromastro. Dopo aver concluso la propria scrittura nel libromastro, il processo sblocca il semaforo, le transazioni nella transaction pool compiono uno shift indietro di SO-BLOCK-SIZE posizioni e viene aggiornata la variabile contenente la quantità di transazioni ancora non processate nella transaction pool.

5.2 Processo utente

Il processo utente, come il processo nodo, definisce e inizializza variabili, oggetti IPC e le struct utili al processo e trova il proprio indice nell'array di PID nella memoria condivisa. Esso esegue un ciclo in cui aggiorna il suo budget con il valore presente nella memoria condivisa, aumentato attraverso un altro ciclo, che conta le quantità ricevute dal processo utente nelle righe del libromastro aggiunte dopo l'ultima iterazione. Se il budget totale è minore o uguale di 2, viene incrementata la variabile limit, che rappresenta i cicli consecutivi in cui il processo non ha potuto inviare nulla poiché con budget troppo basso; se il budget corrente è maggiore, invece, il processo utente inizializza i campi della struct transazione nella coda di messaggi, aggiorna il budget sottraendo il costo di quest'ultima e fa la msgsnd. Se quest'ultima fallisce viene restituito tutto al processo utente e, in seguito, viene eseguita una nanosleep di un valore compreso tra i due passati a runtime. Questo ciclo viene eseguito finché il processo non viene terminato dal master o il valore di limit non è uguale a SO-RETRY, nel qual caso il processo utente muore prematuramente. Una transazione, inoltre, può anche essere inviata su richiesta: lo svolgimento di quest'ultima è uguale ad una transazione generata normalmente, se non che viene generata attraverso il segnale SIGUSR1, il quale assegna alla variabile controllo il PID del processo selezionato. Questa variabile è la prima ad essere controllata nel ciclo d'invio delle transazioni e, dunque, qualora fosse uguale al PID del processo corrente, la struct sarebbe inviata al nodo attraverso la coda di messaggi.

6 Stampa informazioni

Il processo master è deputato alla stampa continua, ogni secondo (per ottenere tale risultato esegue un ciclo while basato sui due valori della struct timespec, invocata con una nanosleep), della situazione dei budget dei processi utente e dei processi nodo. Essi, attraverso i loro codici, aggiornano costantemente i budget presenti nella memoria condivisa MP, alla quale ha un riferimento anche il processo master. Per questo motivo, il master fa delle serie di cicli annidati, che gli permettono di assegnare i valori corretti ai diversi array di cinque elementi, che rappresentano i cinque processi utenti con budget maggiore, i cinque con budget minore, i cinque processi nodo con budget maggiore e i cinque con budget minore. I processi nodo, inoltre, possono stampare "Nodo PID ha inviato transazione a nodo per tp piena" o "Nodo PID ha inviato transazione a master per tp piena", in caso di transaction pool piena, in base al valore di hops.

7 Chiusura della simulazione

La simulazione si può concludere per vari motivi, in base alle combinazioni di argomenti date e alle estrazioni randomiche: tutti gli utenti sono morti, fine tempo simulazione o libromastro pieno. La gestione della chiusura della simulazione si intreccia con la gestione della terminazione dei processi e si basa tutto sull'handler dei segnali, presente in tutti e tre i processi e sul controllo della memoria condivisa contenente le informazioni riguardanti i processi. Nel processo master l'handler gestisce un tipo di segnale: SIGALRM, lanciato quando il timer di SO-SIM-SEC secondi, attivato con l'alarm dal master, arriva a 0 secondi. Se il processo master riceve il segnale SIGALRM assegna 1 alla variabile control, utile per il conteggio degli utenti morti prematuramente. Se il processo figlio invece termina per cause proprie (quindi il processo non è ancora terminato a causa dello scadere del tempo), è settata a 1 la cella riferita al processo nell'array DEAD-USERS, contenuto nella memoria condivisa MP, che determina, dunque, se l'i-esimo processo utente è ancora vivo (se è 0) o se è morto(se a 1). La variabile utentimorti è la somma di tutte le celle dell'array DEAD-USERS, variabile che viene aggiornata ad ogni ciclo di aggiornamento dei valori della memoria condivisa e di stampa, ed è importante poiché, qualora fosse uguale a SO-USERS-NUM, determinerebbe la fine del processo per la morte di tutti i processi utente. Per quanto riguarda i processi nodo, se quest'ultimo riceve una SIGTERM, esso non fa altro che uscire dal ciclo continuo e aggiornare il valore delle transazioni nella transaction pool, non ancora processate nel libromastro. I processi utente, invece, se ricevono una SIGTERM, fanno una exit e terminano la loro esecuzione. Il processo master, infine, gestisce la stampa conclusiva, che fa un resoconto di ciò che è successo all'interno della simulazione: se control è uguale a uno, la conclusione è dovuta alla fine del tempo; se l'indice posizione nel libro mastro è maggiore di SO-REGISTRY-SIZE, la simulazione è terminata poiché si è riempito il lbromastro, mentre se la variabile utentimorti è uguale a SO-USERS-NUM, allora tutti gli utenti sono morti. Dopo la stampa

del motivo di terminazione, viene scritto il numero di blocchi nel libromastro, il numero di utenti morti prematuramente, il budget di tutti gli utenti, di tutti i nodi e il numero di nodi presenti ancora nella transaction pool per ciascun processo nodo, dati ottenuti facendo un ciclo for che scorre tutti gli indici della memoria condivisa MP. Per terminare tutti i processi, il master invoca una kill con SIGTERM e la invia a tutti gli utenti e nodi (ottenendo i PID dalla memoria condivisa MP), rimuove il semaforo con una semctl con parametro IPC-RMID e come ultima cosa esegue una exit, che conclude il processo master e, di conseguenza, la simulazione.