Simulazione di un sistema multithreaded e multiprocesso di un supermercato

Progetto di Laboratorio di Sistemi Operativi

a.a 2019/20

Ferraro Domenico
Matricola 559813
Corso B

Sommario

1 Introduzione

La simulazione del sistema supermercato è realizzata mediante due processi che, come da specifica, implementano rispettivamente l'entità direttore e il supermercato stesso. Il processo che deve essere inizialmente lanciato per l'avvio della simulazione è il processo direttore il quale eseguirà la chiamata di sistema *fork()* per lanciare il processo supermercato (passandogli il path del file di configurazione). Questa azione corrisponde all'apertura del supermercato stesso. I due processi, una volta avviati, stabiliscono una connessione socket AF Unix sulla quale il supermercato comunicherà al direttore le notifiche dei cassieri riguardanti il numero dei clienti in coda e eventuali clienti con zero acquisti che richiedono il permesso di uscita. Il direttore riceve le richieste da parte del supermercato, le elabora e invia una risposta adeguata: nel primo caso potrebbe essere l'apertura o la chiusura di una cassa da lui indicata mentre nel secondo caso può essere il concedere permesso di uscita oppure no. Si tratta quindi di un paradigma client-server in cui il server è il processo direttore e il client è il processo supermercato. Entrambi i processi realizzano le loro funzionalità implementando più thread POSIX.

La comunicazione via socket avviene seguendo un protocollo ben preciso: l'intero messaggio viene preceduto da un cosiddetto header (*enum msg_header_t*) ovvero un intero che indica in maniera inequivocabile a cosa si riferisce il messaggio. In questo modo il ricevente è in grado di interpretare il messaggio e di eseguire un corretto parsing dello stesso limitando la possibilità di errori.

Al termine della simulazione, il sistema scrive sul file di log indicato nel file di configurazione tutte le statistiche da esso accumulate: statistiche per ogni cliente, per ogni cassiere e statistiche generali. Le statistiche per ogni cliente sono ordinate in base al momento di ingresso nel supermercato, quindi la prima statistica si riferisce al primo cliente entrato mentre l'ultima si riferisce all'ultimo cliente entrato. Il programma gestisce le variabili riguardanti il tempo in millisecondi al fine di ottenere una buona precisione ma sul file di log tutti i tempi vengono convertiti e scritti in secondi con al più tre cifre decimali. Lo script di analisi è in grado di poter eseguire il parsing del file, analizzando le statistiche in esso contenute e fornendo un sunto della simulazione sullo standard output. Oltre a stampare le informazioni stabilite dalla specifica del progetto, lo script di analisi fornisce anche il numero di clienti entrati nel supermercato, il numero di clienti usciti senza acquisti oppure che sono stati serviti e il numero totale di prodotti acquistati.

2 Direttore

Il processo direttore implementa le sue funzionalità mediante due thread POSIX: il primo è il thread principale mentre il secondo è un thread adibito alla sola ricezione dei segnali. Il thread principale svolge tutte le mansioni principali come ad esempio il lancio del processo supermercato, la lettura del file di configurazione e la ricezione e l'invio dei messaggi sul canale di comunicazione. Inoltre, si occupa anche di elaborare i messaggi ricevuti, ovvero quando riceve un permesso di uscita risponde sempre concedendolo, mentre quando riceve il numero di clienti in coda in una determinata cassa fa partire l'algoritmo di apertura/chiusura casse. Il direttore ha pieno controllo sullo stato di apertura di ogni singola cassa in quanto può inviare al processo supermercato di aprire o chiudere la cassa da lui indicata. Il thread principale mantiene un array con il numero di clienti in coda per ogni cassa. Non si tratta di un array aggiornato in tempo reale poiché viene aggiornato ogni volta che riceve una notifica dal supermercato oppure quando una cassa viene aperta o chiusa. L'algoritmo, basandosi sui parametri S1 ed S2 contenuti nel file di configurazione, stabilisce quale cassa deve essere aperta e quale deve essere chiusa ed il thread principale invia eventualmente al supermercato la richiesta di apertura/chiusura della cassa.

Il thread principale ed il thread gestore dei segnali comunicano via pipe: una volta ricevuto un segnale, questo viene catturato dal thread gestore dei segnali e inviato via pipe al thread principale. Di conseguenza, il ciclo di vita del thread principale prevede la chiamata della system call *select()* con la quale esamina lo stato dei due descrittori di file, quello relativo alla pipe e quello relativo al socket, garantendo una efficiente gestione dei due canali di comunicazione.

Una volta ricevuto un segnale relativo alla chiusura del supermercato, il direttore lo fa rimbalzare al processo supermercato (ovvero gli invia lo stesso segnale) e rimane in attesa che questo termini prima di terminare a sua volta. I segnali che vengono gestiti in tal senso sono quindi i segnali SIGQUIT e SIGHUP. Inoltre se viene ricevuto un segnale SIGINT questo porta allo stesso esito di quando viene ricevuto il segnale SIGQUIT.

3 Supermercato

Anche il processo supermercato implementa più thread POSIX per realizzare tutte le sue funzionalità. Questo processo modella l'intero supermercato quindi tutti i clienti, la gestione dell'accesso a gruppi, tutti i cassieri e la gestione delle loro code. Lo fa mediante i seguenti thread POSIX: thread principale, thread gestore dei segnali, due thread per ogni cassiere e un pool di thread grande quanto il massimo numero di clienti che può entrare nel supermercato.

Il thread gestore dei segnali, cattura il segnale ricevuto dal processo direttore e lo rimanda via pipe al thread principale il quale lo traduce nel relativo stato di chiusura del supermercato. Il thread principale riceve tramite socket dal direttore le richieste di apertura o chiusura di una cassa e se è concesso o meno ad un determinato cliente di uscire dal supermercato. In tali circostanze, il thread principale traduce in maniera concreta le volontà espresse dal direttore, quindi comunica al thread cassiere di aprire/chiudere la cassa e comunica al relativo thread cliente che gli è concesso uscire o meno. Per una gestione intelligente dei due canali di comunicazione, il ciclo di vita del thread principale prevede la chiamata della system call *select()* con la quale esamina lo stato dei due descrittori di file, quello relativo alla pipe e quello relativo al socket, garantendo efficiente gestione dei due canali di comunicazione. La scrittura sul socket viene fatta in mutua esclusione (proteggendola con una mutex) in quanto clienti e cassieri scrivono sul socket mentre la lettura non prevede race conditions in quanto è il thread principale l'unico lettore.

Il cassiere viene modellato mediante un thread apposito il quale a sua volta genera un thread adibito alle notifiche periodiche del numero di clienti presenti in coda. Per quanto riguarda i clienti, invece, poiché è noto il massimo numero di clienti C che possono essere nel supermercato nello stesso istante, viene implementato un thread pool di C thread riutilizzabili. Ognuno di questi thread rimane in attesa che si possa entrare nel supermercato, svolge le operazioni dell'entità cliente e la sua uscita, e quindi si rimette in attesa di poter entrare nuovamente nel supermercato ma sottoforma di nuovo cliente (a meno che il supermercato non chiuda). Con questa soluzione, l'unico overhead è dato in fase di creazione del thread pool ma riutilizzando intelligentemente i thread viene garantita una ottimizzazione delle risorse a runtime.

Ogni thread è in grado di riconoscere da solo se il supermercato sta chiudendo o meno e svolge autonomamente la sua terminazione. Il thread principale, quindi, si occupa solo di cambiare lo stato del supermercato in base al segnale ricevuto dal processo direttore e di svegliare eventuali thread dormienti su condition variables. Una volta che tutti i thread hanno terminato la loro esecuzione, il thread principale aggrega tutte le statistiche, passa alla scrittura del file di log e termina la sua esecuzione.

3.1 Gestione degli ingressi

Ogni thread cliente rimane in attesa di entrare nel supermercato aspettando su una variabile di condizione. Quando un cliente esce dal supermercato aumenta il contatore dei clienti usciti e se questo contatore è pari ad E sveglia tutti i clienti (*pthread_signal_broadcast*) che aspettano sulla variabile di condizione. I clienti svegliati entrano nel supermercato uno alla volta fino a quando sono stati fatti entrare E clienti. In questo modo, l'uscita del E-esimo cliente provoca l'entrata di altri E clienti. All'inizio, il supermercato consente l'ingresso a C clienti e successivamente si instaura il meccanismo di entrata a gruppi di cui sopra.

3.2 Clienti

Un cliente entrato nel supermercato procede con gli acquisti e il tutto viene simulato rimanendo in attesa per un numero di millisecondi casuale tra 10 e il parametro T e stabilendo come numero di prodotti acquistati un valore casuale maggiore o uguale a zero ma minore del parametro P. I due parametri T e P sono presenti nel file di configurazione. I numeri casuali sono generati attraverso la chiamata della funzione *rand_r* con un seed univoco per ogni thread cliente. Per quanto riguarda le attese, esse vengono realizzate con la chiamata di funzione *nanosleep*. Un cliente con zero prodotti invia una richiesta di uscita al direttore via socket AF Unix e rimane in attesa su una condition variable. Verrà svegliato dal thread principale non appena il direttore invia una risposta sul canale di comunicazione. Se il direttore concede l'uscita, il cliente esce dal supermercato (ed il thread che lo gestiva viene riutilizzato).

Se un cliente acquista un certo numero di prodotti allora valuta tutte le casse aperte e si mette in coda nella migliore. Ad ogni cassa è associata una coda e per via del tempo di servizio di un cassiere, ogni coda potrebbe essere più lenta delle altre. Inoltre, anche il numero di clienti in coda e quanti prodotti ogni cliente possiede influisce su quanto una coda sia più o meno veloce. Sia C il numero di clienti in coda e sia PTOT la somma

dei prodotti che ogni cliente in coda vuole acquistare, ad ogni coda è possibile associare un certo *costo*, ovvero un valore pari a

$$costo = C * tempo\_fisso\_cassiere + tempo\_gestione\_prodotto * PTOT$$

Dove $tempo\_fisso\_cassiere$ è un valore diverso per ogni cassiere nel range *20-80* millisecondi e $tempo\_gestione\_prodotto$ è il tempo di gestione di un singolo prodotto da parte di un cassiere, specificato nel file di configurazione con il parametro KT. La migliore cassa è quindi quella la cui coda ha il minimo *costo* e che, in altri termini, è quella in cui il cliente verrà servito prima.

Mentre un cliente è in coda, diversi eventi possono avvenire: la cassa viene chiusa, il supermercato viene chiuso. Inoltre, il cliente vuole valutare se è più conveniente accodarsi in un'altra cassa aperta (*algoritmo di cambio cassa*). Per gestire queste situazioni, un cliente in coda non rimane totalmente dormiente aspettando di essere servito, ma si risveglia in maniera periodica. In questo modo è in grado di reagire in maniera opportuna ai suddetti eventi. In particolare, quando il cliente svolge *l'algoritmo di cambio cassa*, calcola il costo della sua coda relativo alla sua posizione in coda e si sposta se e solo se esiste un'altra cassa con un costo inferiore. Il costo della sua coda viene calcolato allo stesso modo ma C e PTOT valgono rispettivamente il numero di clienti che verranno serviti prima di lui e la somma dei loro prodotti.

3.3 Cassieri

Al suo avvio, il thread cassiere lancia il suo thread notificatore, ovvero uno speciale thread che si occupa di notificare al direttore ad intervalli regolari via socket AF Unix il numero di clienti in coda nella cassa. Se la cassa viene chiusa, la notifica viene messa in pausa. Questa soluzione prevede di poter gestire i clienti in coda contemporaneamente all'invio di notifiche ed inoltre l'intervallo tra le notifiche risulta essere estremamente vicino se non uguale a quello desiderato.

La coda di un cassiere è una struttura dati dedicata: si tratta di una double linked list di tipo FIFO con aggiunte e rimozioni di complessità O(1) ma alla quale sono associate una variabile che ne indica il costo ed una variabile che indica il numero di clienti in coda. Durante le fasi di aggiunta o rimozione dalla coda, il costo della stessa ed il numero di clienti viene aggiornato. Di conseguenza, per un cliente non in coda, conoscere costo e numero di clienti in coda ha complessità O(1). Nel caso in cui un cliente in coda voglia conoscere il costo della cassa in relazione alla sua posizione, la complessità è pari al numero di clienti che lo precedono in coda (ovvero che verranno serviti prima) in quanto è necessario scorrere la lista partendo dalla posizione del cliente. Gli elementi aggiunti in questa double linked list sono di tipo *client_in_queue_t*, quindi non viene esposta al cassiere la struttura dati del cliente ma solo ciò che veramente è necessario (Information Hiding).

Il cassiere durante il suo ciclo di vita si occupa della gestione della coda a lui associata nel caso in cui la cassa sia aperta, altrimenti rimane dormiente fino a quando la cassa non viene aperta. In caso di cassa aperta, il cassiere preleva dalla coda il cliente successivo, imposta il flag *processing* ad uno per indicare che sta per servire il cliente così che quest'ultimo non cambi cassa. Il cassiere calcola quindi il numero di millisecondi necessari per servire il cliente, esegue la chiamata della funzione *nanosleep*, imposta il flag *served* del cliente ed esegue *pthread_cond_signal* sulla variabile di condizione sulla quale il cliente aspettava di essere servito. Il cassiere esegue la *nanosleep* non avendo nessuna lock e quindi non viene creata congestione.

3.4 Scrittura del log

Dal punto di vista implementativo, particolare attenzione va data all'aggregazione delle statistiche dei clienti. Ogni thread cliente genera una linked list FIFO nella quale sono contenute le statistiche per ogni cliente entrato e uscito dal supermercato mediante questo thread. Un thread cliente gestirà nel suo ciclo di vita un certo numero di clienti. Ogni volta che esce dal supermercato, le statistiche che riguardano il cliente vengono aggiunte in lista con complessita O(1). Come si può intuire, la lista di ogni thread cliente è quindi ordinata in base al momento di ingresso. Il thread principale, quindi, fonde e ordina tutte le liste utilizzando l'algoritmo MergeSort. Una soluzione del tutto motivata dalla natura ordinata delle liste stesse. Un comune difetto di

questo algoritmo è l'utilizzo di spazio extra. Nella soluzione proposta, però, le liste vengono fuse nella prima garantendo spazio extra pari a O(1).

4 Istruzioni

Per avviare la simulazione bisogna lanciare il processo direttore. È possibile specificare il file di configurazione utilizzando l'opzione *-c <path>* dove path è il percorso del file di configurazione desiderato. Se il file di configurazione non viene specificato, viene utilizzato quello di default (*config.txt*) contenuto nella directory principale. Un esempio di lancio della simulazione può essere

./direttore -c path/to/mioconfig.txt

Di seguito è indicata la lista di ogni parametro che deve essere contenuto nel file ed il relativo significato:

- K: numero di cassieri del supermercato.
- KT: tempo di gestione di un singolo prodotto da parte di un cassiere. Espresso in millisecondi
- KA: numero di casse aperte all'apertura del supermercato
- C: numero massimo di clienti che può essere contemporaneamente all'interno del supermercato
- E: quanto è grande il gruppo di clienti che entra nel supermercato
- T: massimo tempo per gli acquisti di un cliente. Espresso in millisecondi
- P: massimo numero di prodotti acquistabili da un cliente
- S: ogni quanti millisecondi il cliente esegue l'algoritmo di cambio cassa
- S1: valore soglia per chiusura di una cassa
- S2: valore soglia per apertura di una cassa
- D: ogni quanto tempo una cassa comunica con il direttore. Espresso in millisecondi
- L: nome del file di log generato dal programma

All'avvio della simulazione il file di configurazione deve indicare tutti i parametri i quali devono anche essere validi ovvero solo valori positivi, E minore di C, KA minore o uguale a K. Se questi criteri non vengono rispettati la simulazione viene abortita in fase di lettura del file di configurazione.

È possibile indicare i parametri in qualsiasi ordine ed è possibile inserire commenti dentro al file precedendoli con il carattere '#'.

5 Makefile

Il progetto include un makefile avente il target *all* per la generazione degli eseguibili del programma supermercato e direttore, *clean* e *cleanall* per ripulire la directory dai file generati come logfiles, file di configurazione dei test e librerie. Tra le librerie utilizzate oltre a quella dei thread POSIX vi sono anche due librerie da me solitamente utilizzate: una libreria di utilities e una libreria per le linked list. Sono presenti i target *test1* e *test2* per il lancio dei rispettivi test. In tal caso viene generato nella directory di lavoro il file di configurazione relativo al test lanciato, se mancante.

In fase di stesura della presente relazione, la simulazione e i test girano senza errori sulla macchina virtuale Xubuntu fornita per il corso e configurata con almeno due cores.