Progetto Sistemi Operativi

Anno 2019/2020

Niccolini Alessandro - Corso A - Matricola 559572 - Data 03/06/2020

Introduzione

Il progetto consiste nella realizzazione di un supermercato diretto da un direttore che ha la facoltà di chiudere, aprire casse e di far entrare e uscire i clienti. Per far partire l’esecuzione del progetto utilizziamo il makefile (make test2) che provvederà a compilare tutti i file.c secondo l’ordine delle dipendenze, a creare una libreria, a lanciare l’eseguibile e successivamente a lo script che permetterà di visualizzare un riassunto di quello che è stato fatto durante l’attività di cassieri e clienti. All’avvio del supermercato viene letto un file di configurazione e vengono presi i parametri che andranno a definire le varie soglie e valori (K, C, E, T, P, ecc). Vengono aperte un certo numero di casse stabilite nel file di configurazione, e vengono fatti entrare C clienti, personalmente ho scelto di aprire tutte le casse al momento di apertura del supermercato in moda da permettere di smaltire e distribuire tutti i clienti che entrano nello stesso momento. Una volta che il valore di clienti nel supermercato scende sotto C – E vengono fatti entrare altri E clienti. Alla ricezione di un segnale (SIGHUP, SIGQUIT) il direttore attiva l’handler corrispondente ed agisce di conseguenza. Nel caso di SIGHUP vengono finiti di servire tutti i clienti già all’interno del supermercato e successivamente si procede a chiudere le casse rimaste aperte ed a terminare il direttore. Mentre nel caso del segnale SIGQUIT vengono fatti uscire immediatamente tutti i clienti, chiudo le casse e faccio terminare il direttore. Dopodiché salverò i dati prodotti durante l’esecuzione del supermercato nel file di log finale (specificato nel file di configurazione), solo nel caso della chiusura “graduale” salverò le info dei cassieri, perché nel caso della chiusura immediata, avendoli cancellati mentre svolgevano del lavoro potrei ritrovarmi dati inconsistenti. Mentre per quanto riguarda i clienti, le informazioni vengono salvate ogni volta che termina uno. Dopo verrà lanciato uno script che si occuperà di fare il parsing del file di log ed a stampare ordinatamente tutti i dati. Per questo progetto ho usato alcune strutture dati che sono in comune a più file, perciò, l’ho definite in utility.h e quando ne ho bisogno lo includo.

MakeFile

Il file dependency system di Unix serve ad automatizzare l’aggiornamento in modo corretto di più file con delle dipendenze, permettendo quindi di mantenere il sistema in uno stato consistente e di facilitare la creazione dei file oggetto, della libreria e del file eseguibile. Inoltre, permette di facilitare la pulizia dei file non più necessari dopo la terminazione del programma. Per facilitare e ottimizzare il codice ho creato una libreria dinamica, che è agganciata al programma in due fasi. A tempo di compilazione quando il linker verifica che siano definiti tutti i simboli e nella fase di caricamento dove il loader carica le librerie necessarie agganciate al programma.

Supermercato

Il supermercato si occupa caricare le informazioni iniziali dal file di configurazione, di inizializzare le strutture che userà e di creare il thread direttore. Il quale impiegherà le informazioni che gli sono state passate per la creazione di thread clienti e thread cassieri. Il supermercato inoltre aspetterà la terminazione del direttore e dopo caricherà le informazioni riguardanti le statistiche dei clienti e dei cassieri raccolte durante la fase di esecuzione del supermercato nel file di log, specificato nel file di configurazione.

Direttore

Il direttore è il fulcro di tutto il progetto, è colui che gestisce l’entrata e uscita dei clienti nel supermercato, l’apertura o la chiusura delle casse, si occupa di ricevere e gestire il segnale di chiusura (SIGHUP o SIGQUIT) e di spostare i clienti in altre casse se quella chiude. Per fare questo sfrutta diverse strutture che gli permettono di immagazzinare informazioni, e le passa a chi ne ha bisogno. Inoltre, prima del supermercato è l’ultimo thread che termina, infatti deve aspettare che clienti e cassieri abbiano terminato il proprio lavoro. Nel caso della chiusura graduale il direttore oltre a non far entrare più clienti si mette ad aspettare quelli già all’interno del supermercato, successivamente chiude le casse aperte e termina. Mentre nel caso della chiusura immediata fa uscire tutti i clienti e chiude le casse, dopodiché termina. All’inizio il direttore apre un numero di casse definite nel file di configurazione e fa entrare i primi clienti, dopo entra in un ciclo nel quale quando viene notificato dai clienti che il numero complessivo dei tali è minore di una certa soglia allora crea e fa entrare altri thread clienti. Questo ciclo continua finché non riceve il segnale di chiusura. Prima di terminare libera la memoria allocata che non serve più.

Cassiere

Il cassiere è inizializzato mediante una funzione che assegna i valori alle variabili presenti nella struttura di supporto, subito dopo viene fatto partire il thread cassiere che oltre a bloccare i segnali non desiderati, fa partire un thread di supporto al cassiere che provvederà a informare il thread direttore sul numero di clienti in coda nella cassa, a intervalli di tempo regolari stabiliti nel file di configurazione. Adesso il cassiere inizia il suo ciclo per servire i clienti, se non ci sono aspetta che qualcuno si metta in fila, altrimenti serve il cliente. Il cliente informa il cassiere del proprio tempo totale di attesa in coda, in modo che il cassiere possa calcolare alcuni parametri relativi al cliente che serviranno per collezionare i dati di statistica ed aggiorna le variabili relative al proprio lavoro. Questo ciclo verrà ripetuto finché il cassiere rimane aperto. Una volta che riceve dal direttore l’ordine di chiudere aggiorna i dati raccolti durante il tempo di servizio e termina.

Cliente

Il Cliente inizia la fase di inizializzazione non appena viene fatto entrare nel supermercato, anche lui blocca i segnali non desiderati, dopodiché passa un tempo variabile all’interno per comprare prodotti, compra un certo numero di prodotti casuale (ma in un certo range che dipende da un parametro di configurazione), se il numero di prodotti è 0 allora non si mette in coda in una cassa ma aggiorna i suoi dati che vengono poi raccolti dal direttore e chiede il permesso di uscire. Mentre nel caso di prodotti maggiore di 0, si mette in fila in una cassa random tra quelle aperte ad aspettare il proprio turno. Quando è il suo turno calcola il tempo trascorso nella coda ed informa il cassiere, che dovrà calcolare alcuni dati utili al fine di raccogliere informazioni sul cliente, non appena avrà terminato informerà quest’ultimo. Alla fine, comunicherà i dati raccolti durante la sua permanenza nel supermercato al direttore. Se il cliente è l’ultimo nel supermercato oppure vede che i clienti sono certo una certa soglia, allora informa il direttore e termina, altrimenti termina semplicemente.

Script

Lo script Bash analisi.sh permette di ottenere un riassunto delle informazioni raccolte dal programma durante la sua esecuzione. Tramite il parsing del file di log è possibile visualizzare i dati in maniera ordinata e chiara di quello che è stato fatto durante il tempo in cui sono stati all’interno del supermercato dei cassieri e dei clienti.

Scelte implementative

Per quanto riguarda le scelte lasciate a discrezione nostra, ad esempio nel caso di quale cassa aprire ho scelto di usare l’indice minore libero, ossia se il numero di casse aperte è inferiore al numero massimo apro la cassa con indice minore che è chiusa. Mentre per quale cassa chiudere la stessa cosa ma con indice maggiore, ossia, se il numero di casse aperte è maggiore di 1 chiudo la cassa con indice maggiore tra quelle aperte. Quando devo chiudere una cassa, la scelta per come assegnare i clienti nelle altre code dei cassieri aperti è random, in base alle casse aperte. La soglia S1 stabilisce quante casse con al più un cliente ci devono essere perché una cassa venga chiusa, mentre la soglia S2 stabilisce quante clienti in coda in una cassa ci devono essere perché venga aperta una nuova cassa.

Pthread

Sono uno standard di thread POSIX che nel corso del tempo hanno continuato ad evolversi.

Nelle moderne macchine multi-core i pthread sono ideali per la programmazione parallela. Sono adatti nei casi in cui:

- del lavoro può essere eseguito da più tasks contemporaneamente

- aspettare operazioni di I/O

- uso di cicli della CPU in alcuni posti ma non in altri

- c’è bisogno di rispondere ad eventi asincroni

I pthread, condividono spazi di memoria, ma hanno anche spazzi privati. Quelli che condividono tuttavia devono essere accessi in modo “sicuro”, ovvero in maniera tale da garantire l’accesso a quella variabile ad un thread alla volta, per prevenire situazioni di inconsistenza dei dati. Durante il mio progetto ho utilizzato molto i pthread in quanto tramite lock riesco a garantire l’accesso ad un thread alla volta a quelle variabili che sono condivise, oppure a rendere atomiche certe operazione come aggiungere i clienti nella coda. Senza le quali andrei a finire in situazioni di Race condition, per cui nelle sezioni critiche del mio codice ho usato la mutua esclusione per prevenire condizioni anomale. Ed ho usato le variabili di condizione quando un thread doveva aspettare che un altro svolgesse un’operazione.

Standard I/O library

Le funzioni che fanno parte della libraria standard di I/O forniscono la bufferizzazione in spazio utente, sono più veloci ed efficienti nel caso in cui andiamo a leggere o a scrivere pochi dati alla volta.

Segnali

I segnali sono delle interruzioni software che comunicano al processo il verificarsi di un evento. Sono innumerevoli, ed ognuno corrisponde ad un evento in particolare. Quando arriva un segnale è possibile svolgere diverse azioni, tra cui:

- ignorarlo

- lanciare una funzione creata apposta per gestire quel segnale (signal handler), oppure usare la funzione di default del kernel

Lo standard POSIX fornisce una serie di segnali riconosciuti in tutti i sistemi adatti.

Nel caso di questo progetto avendo a che fare con molti thread, ma dovendo arrivare il segnale solo al direttore ho mascherato il segnale in tutti gli altri thread e creato un handler apposito per gestire il segnale da parte del direttore.

Regole per usare il Makefile

Per usare il makefile digitare nel terminale aperto nella directory dove è presente il makefile:

bash:~$ make test2 (per avviare la fase di test)

bash:~$ make clean (per rimuovere i file creati durante il test)