RELAZIONE PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI

a.a. 2019/2020

Studente:
Luigi Gesuele
Matricola: 562376
Corso: A

Docenti:
G. Prencipe
M. A. Bonuccelli

Scelte progettuali

La versione semplificata del progetto richiedeva l'implementazione di un supermercato come unico processo multithread, per cui tutte le strutture dati utilizzate sono definite globalmente e quindi condivise tra tutti i thread, che vi accedono in mutua esclusione mediante mutex e condition variables. Questo non è stato fatto invece per le variabili relative ai due segnali da gestire, SIGHUP e SIGQUIT, poiché le due variabili globali associate a questi segnali sono state definite di tipo sig_atomic_t, ed essendo gestite quindi con accessi atomici non erano necessarie le mutex o le condition variables.

Come richiesto dal testo, il supermercato deve essere aperto da un Direttore che farà entrare inizialmente tutti i clienti concorrentemente e aprirà un numero predefinito di casse inizialmente. Inoltre, il direttore deve poter controllare anche le entrate e uscite dei clienti stessi, e deve essere notificato costantemente sulla lunghezza delle code di ciascuna cassa per poter decidere quali aprire e chiudere. Sulla base di queste richieste, ho pensato di crearmi un thread **"Direttore"** che si occupasse di gestire sia le casse che i clienti mediante due sottothread.

Il sottothread **"Direttore Casse"** si occupa quindi di avviare i K thread Casse assieme ai K thread Timer, che servono esclusivamente a inviare a intervalli di tempo predeterminati al Direttore le dimensioni delle code di tutte le casse, come richiesto dalla specifica. Quindi il Direttore Casse rimane in attesa di ricevere tutte le informazioni delle casse aperte per poi effettuare i controlli S1 e S2 della specifica, per poter decidere quali casse aprire o chiudere. Quindi il thread Direttore Casse, alla ricezione di un segnale di SIGQUIT o SIGHUP deve attendere il termine di tutti i thread "Casse" e "Timer".
Il sottothread **"Direttore Clienti"** invece si occupa di avviare i C thread Clienti, dopodiché si mette in attesa di dover gestire i clienti uscenti dal supermercato, per poterne poi far rientrare E quando il numero di clienti all'interno del supermercato raggiungerà la soglia C-E. Per semplicità ho pensato di creare una coda gestita dal Direttore Clienti, che sarà riempita via via che un Cliente termina gli acquisti presso una qualunque Cassa oppure quando un Cliente decide di non acquistare nulla. Anche qui, alla ricezione di un segnale il thread Direttore Clienti attende la terminazione di tutti i thread Clienti e termina.

Il thread **"Cassa"** coincide invece con la figura del cassiere. Il thread resta in attesa che la cassa venga aperta, e dopodiché resta in attesa che la sua coda sia non vuota. Nel primo ciclo ho inoltre aggiunto un controllo sulla dimensione della coda, in quanto se quest'ultima risultasse piena vorrebbe dire che è stata appena chiusa e quindi i Clienti in coda devono cambiare cassa. Se invece la coda di una cassa aperta è non vuota, il cassiere attende, prima di servire ogni cliente, un tempo fisso scelto casualmente con la **rand_r** tra **T_MIN_CASS** e **T_MAX_CASS.** Solo dopo che questo tempo è trascorso la Cassa inizia a servire il Cliente in testa alla coda relativa ad essa, così da permettere a eventuali clienti di accodarsi. Questo perché ho definito un array di K code globale, ciascuna relativa ad ogni cassa, per cui si ha bisogno di un accesso in mutua esclusione. Per ogni cliente in coda, il cassiere passa i prodotti di quest'ultimo attendendo il tempo fisso per passare un prodotto moltiplicato il numero di prodotti acquistati, e oltre ad aggiornare il tempo in coda del cliente che sta servendo, aggiorna anche i dati di clienti serviti e numero di prodotti elaborati. Questo perché il thread Cassa si occupa anche di scrivere nel file di log le informazioni relative a ciascun cliente via via che li serve. Inoltre, il thread cassa scrive nel file di log le informazioni sulla cassa stessa quando questa viene chiusa. Vengono inoltre via via aggiornati anche le statistiche generali del supermercato.

Il thread **"Cliente"** coincide con la figura del cliente. Una volta avviato, viene scelto casualmente sempre con la funzione **rand_r** sia un numero di prodotti da acquistare, sia un tempo che viene impiegato dal cliente stesso per effettuare gli acquisti. Se il numero di prodotti generato casualmente è 0, allora il cliente va direttamente nella coda del Direttore Clienti, altrimenti sceglie a caso una cassa aperta e ci si mette in coda. Infine il thread Cliente termina.

Il thread **"TimerCassa"** viene avviato insieme ad ogni thread Cassa, poiché per ogni cassa il direttore deve ricevere informazioni in maniera asincrona. Se la cassa è chiusa, il timer resta in attesa, altrimenti attende il tempo S del file di configurazione e poi scrive in un array definito globalmente le informazioni relative alla cassa associato ad esso.

Il controllo da parte del thread Direttore Casse sullo stato delle code di ciascuna cassa, per semplicità lo ho scritto sotto forma di due funzioni a se stanti, chiamate S1() e S2(), per una migliore leggibilità del codice. Inoltre ho aggiunto nel controllo di S1 che vi siano almeno 2 casse aperte, altrimenti verrebbero chiuse tutte le casse, e per il controllo di S2 ho aggiunto che vi siano al più K-1 casse aperte. Il Direttore Casse resta quindi in attesa di ricevere tutte le informazioni solamente dalle casse aperte, per cui nelle posizioni relative alle casse chiuse ho inserito il valore -2, mentre nelle casse aperte da cui non sono ancora state ricevute informazioni ho inserito il valore -1. Ho inoltre ipotizzato che, alla ricezione di un qualsiasi segnale per la chiusura, questo controllo di apertura/chiusura casse fosse superfluo poiché, in caso di SIGHUP occorreva bloccare l'ingresso di nuovi clienti e finire di servire i clienti attuali con i soli cassieri già aperti, mentre in caso di SIGQUIT andavano finiti di servire solo i Clienti in testa e tutti gli altri andavano fatti uscire. Alla ricezione di un segnale, le casse vengono chiuse tutte, così si comporteranno in modo tale da svuotare tutte le code da tutti i clienti e poi terminano, stampando sul logfile.
Nell'handler dei segnali setto la variabile relativa ad esso a 1, ed eseguo la funzione wakeUp() che permette di svegliare eventuali thread "dormienti"(con signal e broadcast) per poter uscire dai rispettivi cicli e agire a seconda del segnale ricevuto.

Strutture dati

I dati relativi ai cassieri e ai clienti sono memorizzati in due **struct**, **client** e **cassiere.** Allo stesso modo, anche i dati per le statistiche generali del supermercato sono salvati nella **struct supermercato.** Infine, i dati del file di configurazione sono letti nel main e poi inseriti in una **struct config** globale chiamata cfg. Le varie code invece sono state implementate analogamente alle code viste nel corso di Algoritmica e Laboratorio, ovviamente riadattate a contenere elementi di tipo client, ed ho aggiunto una funzione ausiliaria che mi permettesse di aggiornare le informazioni relative a ciascun cliente nella coda (come ad esempio tempo in coda totale per ogni cliente ecc…). Ho inoltre creato due header file, **client.h** e **cassiere.h,** che contengono al loro interno le struct relative a cliente e cassiere e funzioni ausiliarie per la generazione casuale di prodotti, tempo per acquisto dei prodotti ecc…
Ho inoltre creato un ulteriore header file, **error.h,** contenente le macro con le definizioni di funzioni per la gestione degli errori, come visto a lezione.
Mi sono inoltre scritto due funzioni, nel file supermercato.c, chiamate **initAll()** e **cleanAll(),** che servono rispettivamente per inizializzare le varie strutture dati, mutex e variabili di condizione, e per liberare lo heap dalle allocazioni di memoria. Sono presenti nel codice anche due funzioni ausiliarie per contare il numero di casse aperte e restituire un array di casse aperte, poiché potessi poi riutilizzarle. Il file di log contiene inoltre anche la data di apertura e di chiusura del supermercato, e ad ogni avvio del supermercato, elimina il file se presente, per evitare di generare un log errato, in quanto questo file viene scritto in modalità append.

Come usare il programma

Per utilizzare il programma è possibile crearne l'eseguibile eseguendo il comando "**make**" sul terminale. Dopodiché ci sono due possibili esecuzioni: "**make test**" e "**make clean**". Il primo serve per effettuare il test richiesto dalla specifica, che andrà quindi a prendere in input il file di configurazione "config.txt" che si trova nella cartella "config". Alla fine dell'esecuzione, è possibile eseguire "**make clean**" per pulire eventuali file superflui generati durante l'esecuzione del programma. Eseguendo make test, viene eseguito il programma principale con i parametri:
K=6 //numero casse totali
C=50 //numero clienti totali
E=3 //clienti che devono uscire per farne rientrare altri E
T=200 //tempo massimo che ciascun cliente può impiegare nell'acquisto
P=100 //numero massimo di prodotti acquistabile per ciascun utente
S=20 // tempo che un cassiere impiega per passare un prodotto
S1=2 //parametro da rispettare per la chiusura delle casse
S2=10 //parametro da rispettare per l'apertura delle casse
t_info=20 //intervallo di tempo per cui ogni cassa informa il direttore sullo stato della propria coda
c_iniz=1 //numero di casse aperte inizialmente
log=logfile.log //nome del file di log sul quale viene scritto il sunto della situazione

Al termine dei 25 secondi, viene mandato un segnale di SIGHUP al programma, quindi gli eventuali clienti all'interno del supermercato vengono processati ed infine esso termina. Al termine, il makefile esegue lo script analisi.sh, dopo averne cambiato i permessi, che effettua il parsing del file di log sul terminale, sullo standard output.

Configurazione personalizzata

È inoltre possibile utilizzare un file di configurazione personalizzato, affinché questo sia di un formato consono. Per farlo, occorre scriverlo nello stesso format della struttura sopra descritta, ed infine eseguire il comando "**make**" per compilare il programma e generare l'eseguibile, ed infine eseguire da terminale:

make personal conf=pathToFile.format

dove pathToFile è il nome o il percorso del file di configurazione scelto e format è l'estensione di quest'ultimo.
Se si vuole passare un diverso tempo (in secondi) oltre che un file di configurazione diverso da quello di default, si può eseguire invece, dopo aver generato l'eseguibile con make:

make personalTime conf=pathToFile.format time=Xs

dove pathToFile è il nome o il percorso del file di configurazione scelto e format è l'estensione di quest'ultimo, mentre X è il numero di secondi di apertura del supermercato (la s va inclusa per indicare che l'unità di misura è il secondo).