UNIVERSITÀ DI PISA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Relazione del progetto del corso di Sistemi Operativi e Laboratorio

Studente: Lorenzo Massagli
Matricola: 579418
Corso: A

Insegnanti:

Prof. Giuseppe Prencipe
Prof. Maurizio Angelo Bonuccelli

Anno Accademico 2019/2020

Capitolo 1 – Progettazione e Implementazione

1.1 Dettagli progettuali

La fase di progettazione è iniziata con scegliere in quale modo implementare la sincronizzazione e passaggio di valori tra i vari thread nel programma.
Le scelte effettuate sono state le seguenti:
1) Sincronizzazione: Mutex e condition variable per le variabili globali condivise sia in lettura che scrittura dai vari thread. Le variabili globali collegate ai segnali SIGHUP e SIGQUIT sono state invece dichiarate volatile sig_atomic_t essendo delle variabili "booleane" e facilmente controllabili con accesso atomico ad esse.
2) Passaggio di valori: La scelta è stata guidata dalla specifica del progetto, infatti nella versione "ridotta" si poteva scegliere di implementare il direttore come thread interno al processo supermercato. E' stata quindi sfruttata la proprietà dei thread, cioè che tutti i thread condividono lo spazio di memoria del processo che li ha creati. Infatti il passaggio di valori è stato effettuato tramite le variabili globali con accesso in mutua esclusione.

Come ulteriori scelte progettuali, l'uscita e l'entrata dei vari clienti all'interno del supermercato è controllata dal direttore, che darà il consenso di entrata ed uscita nel supermercato ai vari thread cliente . I clienti all'interno del supermercato vengono fatti entrare SOLO E SOLTANTO se sono usciti E clienti dal supermercato. E' stata progetta l'uscita dei vari clienti come una vera e propria coda di uscita dal supermercato in modo da controllare accessi e uscite.

Il direttore è stato suddiviso in due sottothread:
- Un sottothread controlla le casse (Apertura e chiusura) e riceve le varie informazioni sulla lunghezza delle code dai cassieri.
- Un sottothread controlla l'entrata e l'uscita dei clienti dal supermercato.
Attenzione: Sia i thread cliente che i cassieri sono creati dal direttore.

Ogni qualvolta il thread direttore che controlla le casse viene svegliato, controllerà quali casse aprire o quali casse chiudere, non entrambe le azioni nello stesso momento. Questo è stato fatto per evitare di andare a chiudere una cassa, per poi doverla riaprire immediatamente dopo. Avremo così degli intervalli più ampi dove il direttore potrà analizzare la situazione alle casse, e decidere se deve aprire o chiudere una cassa.

1.2 Dettagli di implementazione

Come detto nella fase progettuale, la sincronizzazione e passaggio di valori è stato completamente affidato a variabili globali controllate da mutex e condition variable per garantire la mutua esclusione.
Infatti anche le code (queue) sono gestite tramite mutex e condition variable, uno per ogni cassa, per assicurare l'accesso in mutua esclusione da parte dei cassieri e clienti.
Le casse quando vengono avviate, avviano a loro volta una thread "clock" il quale, fin quando la cassa è aperta, invierà periodicamente le informazioni sulla lunghezza della cassa al thread direttore. Il direttore aspetterà che tutte le casse aperte abbiano mandato un aggiornamento della lunghezza della cassa, per poi decidere quali casse chiudere ed aprire.
In caso di apertura di una cassa, la cassa aperta riprende i valori precedenti (Se era già stata aperta) e continua ad aggiornarli con i nuovi.
In caso di chiusura, aspetta che il thread clock termini, per poi terminare a sua volta. Prima di terminare va a svuotare la coda di attesa e fare un broadcast ai clienti in attesa, per poter permettere loro di cambiare cassa.
I dati del file di log, vengono aggiornati man mano che terminano i vari clienti con i dati raccolti durante la loro esecuzione. Infine vengono aggiunti i dati delle casse e i dati generali del supermercato.
Per prendere il tempo di coda e di esecuzione dei vari clienti/cassieri è stata usata la funzione clock_gettime che permette di fare un timestamp nel momento in cui viene chiamato il metodo. Quindi viene fatto un timestamp all'inizio e alla fine, per poi poter calcolare quanto tempo è passato effettivamente.
I segnali di SIGHUP e SIGQUIT quando ricevuti dal signal handler, esso aggiorna le variabili globali che indicano che c'è stato uno di questi due segnali.

In caso di **SIGQUIT** i thread eseguono le seguenti operazioni:
- Il thread direttore che gestisce i cassieri, termina la sua recezione sulla lunghezza delle code essendo che il supermercato sta chiudendo e aspetta che tutti i thread cassieri attivi terminino.
- Il thread direttore che gestisce i clienti, fa uscire tutti i clienti che man mano arrivano all'uscita. Quando i clienti all'interno del supermercato sono 0, termina. (Non fa entrare altri clienti).
- Il thread cassiere finisce di servire il cliente, se ne stava servendo uno, libera le code e aspetta che il thread clock finisca, per poi terminare.
- Il thread clock termina appena possibile, svegliando il thread direttore che gestisce i cassieri.
- Il thread cliente va direttamente all'uscita, in modo da terminare, senza passare dalle casse.

In caso di **SIGHUP** i thread eseguono le seguenti operazioni:
- Il thread direttore che gestisce i cassieri, termina la sua recezione sulla lunghezza delle code essendo che il supermercato sta chiudendo e aspetta che tutti i thread cassieri attivi terminino.
- Il thread direttore che gestisce i clienti, fa uscire tutti i clienti che man mano arrivano all'uscita. Quando i clienti all'interno del supermercato sono 0, manda un signal alle casse, il quale le avverte che non ci sono più clienti all'interno del supermercato, e conseguentemente, termina. (Non fa entrare altri clienti).
- Il thread cassiere serve i vari clienti che arrivano in coda, fin quando non restano 0 clienti all'interno del supermercato, poi termina.
- Il thread clock termina appena possibile, svegliando il thread direttore che gestisce i cassieri.
- Il thread cliente va alle casse per acquistare i prodotti ed una volta fatta la coda, attende il consenso del direttore per uscire. Poi termina.

Tutti i dati dei cassieri e clienti sono contenuti in delle struct.
Il file del configurazione viene letto dal processo supermercato e inserito in una struct globale di configurazione, che conterrà tutti i parametri definiti.

Il file di configurazione è nella cartella testfile e deve chiamarsi "config.txt".

Capitolo 2 – Come usare il programma e configurarlo

2.1 Come usare il programma

Fare il comando "**make**" da terminale per creare il file .o
Comandi disponibili:
1) **make test** → Esegue il programma con i parametri indicati nel file di configurazione contenuto in ./testfile/config.txt
2) **make clean** → Per ripulire la cartella da file create dall'esecuzione del programma

TIP: Si può anche eseguire il programma in debug mode (Modalità che va a descrivere ogni singolo passaggio che il programma fa) andando a modificare il Makefile alla riga "$(CC) ./supermarket.c $ (FLAGS) -o $@" cambiandola con " $(CC) -g $(DEBUG) ./supermarket.c $(FLAGS) -o $@"

2.2 Modificare i parametri di esecuzione del programma

Per modificare i parametri di input del programma, bisogna andare a modificare il file ./testfile/config.txt dove i parametri indicano:
K → Numero di casse
C → Numero di clienti iniziali
E → Numero di clienti che devo aspettare che escano, prima di farne entrare altri
T → Minimo tempo di acquisto di un cliente
P → Numero di prodotti massimo per ogni cliente
S → Secondi per passare un prodotto da parte del cassiere
S1 → Se ci sono S1 code con al più 1 cliente → Chiudi una di quelle casse
S2 → Se c'è una cassa con più di S2 clienti in coda → Apri una nuova cassa
smopen → Numero di casse aperte all'apertura del supermercato
directornews → Tempo che ogni cassiere aspetta prima di inviare al direttore il numero di clienti in coda

2.3 File di analisi.sh, file di log e makefile

Il **makefile** contiene le operazioni di clean e test come già preannunciato in precedenza, inoltre quando viene chiamato il comando “make” deve andare a creare il file .o del progetto supermercato.

Il **file di log** ha estensione .log e deve contenere i dati di ogni cliente, ogni casse e del supermercato in generale.
I dati di ogni cliente vengono inseriti singolarmente da essi quando terminano.
I dati delle casse e del supermercato vengono inseriti dal processo supermercato prima di terminare.

Il file di **analisi.sh** viene eseguito a terminazione del processo supermercato, e va a fare un parsing delle informazioni scritte nel file di log stampandole su standard output.
Quindi stamperà su stdout i dati dei vari clienti e delle casse.