Relazione Progetto Sistemi Operativi

Lorenzo Angeli - Corso A - 539036

2019/2020

1 Supermercato

L'architettura del supermercato è costituita da tre componenti (o thread) principali: Cliente, Cassiere e Direttore.

1.1 Cliente

Il cliente viene creato in modalità detached dal main (*supermercato.c*). Il meccanismo di ricerca della cassa aperta dove potersi mettere in coda per pagare, avviene in modo randomico. I clienti con 0 prodotti acquistati non vengono inseriti nella coda ma attendono il permesso dal direttore per poter uscire dal supermercato (vedi paragrafo 1.3.2).

```
elemcassa* lista;
static pthread_mutex_t locklista=PTHREAD_MUTEX_INITIALIZER;
```

Ho utilizzato una lista *lista* di interi (presente nella libreria *dati*) dove poter salvare gli indici delle casse aperte utilizzando una variabile di mutex *locklista* per eseguire le varie operazioni in mutua esclusione. La ricerca di una cassa aperta avviene in questo modo:

1. Salvo all'interno di una variabile *dimlista* il numero degli elementi presenti all'interno della lista.
2. Genero un valore casuale *tmp* compreso tra [0, *dimlista*-1]
3. Cerco l'elemento che corrisponde all'indice *tmp* nella lista, esso corrisponderà alla cassa in cui andrò ad inserire il cliente.

Terminato l'inserimento in coda alla cassa, il thread cliente viene fatto terminare. A questo punto sarà il cassiere ad occuparsi della gestione del cliente.

1.2 Cassiere

Il cassiere è stato implementato con attesa tramite join per favorire la gestione di terminazione del programma. Ogni cassiere ha la propria lock e variabile di condizione.

```
pthread_mutex_t* mutex;
pthread_cond_t* cond;
```

La variabile *mutex[i]* serve per avere mutua esclusione sulla modifica delle strutture dati associate alla cassa i, mentre *cond* è una variabile di condizione che ha lo scopo di evitare attesa attiva nel ciclo while che controlla lo stato della cassa e la presenza di clienti in coda. Infatti, nel caso di cassa chiusa o di cassa aperta senza clienti in coda, il thread cassiere i viene fatto sostare nella coda della variabile di condizione *cond* per poi essere riattivato dal cliente tramite signal. Terminata l'operazione di rimozione del cliente dalla cassa tramite funzione *rimuovi* della libreria *dati* può essere servito il cliente. Vediamo adesso quali sono le strutture dati più importanti utilizzate dal Cassiere di cui fanno uso anche le funzioni della libreria *dati*:

1. int* codaclienti: array di K posizioni in cui salvo il numero di clienti in coda, dove ogni posizione dell'array corrisponde ad una cassa ben precisa.
2. int* statocassa: array di K posizioni che indica se la cassa i è aperta (vale 1) o chiusa (vale 0).

1.3 Direttore

Il direttore è unico e principalmente si occupa di mantenere consistenti le informazioni sulle casse e di gestire i clienti che vogliono uscire dal supermercato senza aver acquistato nessun prodotto. La politica di apertura/chiusura casse rispetta i requisiti del progetto. Utilizza però una piccola variazione per evitare che il direttore causi una apertura di una cassa seguita dall'immediata chiusura della stessa o di un altra cassa. E' quindi possibile aprire sempre nuove casse (se le condizioni lo permettono). Se però nell'ultimo istante è stata chiusa una cassa allora non è possibile aprirne una nuova.

1.3.1 Comunicazione tra Direttore e Cassieri

Vediamo di seguito il codice utilizzato per gestire la comunicazione dei cassieri nei confronti del direttore per informarlo sul numero dei clienti in coda alle casse:

```
pthread_mutex_t locknotify=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t attendinotify=PTHREAD_COND_INITIALIZER;
//array di bit (0/1)
int* notifica;
int* cassenotify;
int* clientinotify;
```

Per la comunicazione tra cassieri e direttore si utilizzano K thread di supporto creati in modalità detached, uno per cassiere. Ognuno di questo thread i scrive le proprie informazioni riguardanti la cassa corrispondente dentro due array di interi (*cassenotify* e *clientinotify*) ad intervalli regolari e settano a 1 *notifica[i]* che ha il compito di capire quale cassa deve ancora inviare le proprie informazioni. *cassenotify* contiene le informazioni sugli stati delle casse mentre *clientinotify* contiene le informazioni sul numero di clienti in coda alle casse. Il direttore legge le informazioni da questi due array quando ognuno dei cassieri ha comunicato le proprie informazioni e se ne accorge quando *notifica* ha ogni cella settata a 1. A questo punto il direttore legge i dati e li salva dentro due array di supporto. Per mantenere la mutua esclusione di queste strutture viene utilizzata una lock *locknotify* e una variabile di condizione *attendinotify* con lo scopo di evitare l'attesa attiva da parte del direttore nel ciclo while che controlla se ogni cassiere ha inviato i propri dati.

1.3.2 Gestione uscita dei clienti con 0 prodotti

Vediamo di seguito il codice utilizzato per gestire l'uscita dei clienti senza acquisti e successivamente commentiamolo.

```
int permesso;
pthread_mutex_t lockuscita=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condcliente=PTHREAD_COND_INITIALIZER;
```

Per gestire i clienti che devono chiedere l'autorizzazione al direttore per poter uscire che non hanno acquistato nessun prodotto si utilizzano: una variabile intera (flag) *permesso* (che vale 1 se il direttore ha dato il via libera ai clienti di uscire e 0 altrimenti), una lock *lockuscita* e una variabile di condizione *condcliente*. *lockuscita* serve per mantenere la mutua esclusione della variabile *permesso* mentre nella coda della variabile di condizione *condcliente* si bloccano i thread clienti con 0 acquisti in attesa del permesso da parte del direttore di poter uscire.

2 Librerie

Tutte le librerie sono memorizzate nella directory *lib*. Descriviamo di seguito le librerie utilizzate dalle componenti principali.

2.1 dati

La libreria *dati* contiene due strutture dati: un array di liste *coda* e una semplice linked list *lista*. Ogni posizione dell'array *coda* fa riferimento a una cassa e ogni cassa contiene la lista dei clienti in coda. La lista *lista* invece contiene l'indice delle casse aperte. Le principali funzioni di *coda* sono:

- inserisci: inserisce il cliente in fondo alla lista di una delle casse aperte. L'inserimento non avviene in maniera casuale ma deve essere specificata la cassa dove voler inserire il dato.
- rimuovi: rimuove il primo elemento inserito dalla cassa specificata.
- opencassa: apre la cassa passata come argomento se risulta chiusa.

- closecassa: chiude la cassa passata come argomento se aperta. La chiusura di una cassa sposta ogni elemento della cassa chiusa in una delle casse aperte.

Le principali funzioni di *lista* sono:

- inseriscicassa:inserisce l'indice della cassa passata come argomento in coda alla lista.
- rimuovicassa: rimuove l'indice della cassa passata come argomento dalla lista.
- generacassa: restituisce casualmente l'indice di una delle casse presenti all'interno della lista.

2.2 parsing

La libreria parsing serve per effettuare la lettura del file *config.txt* ed estrapolarne le informazioni in modo che possano essere salvate in alcune variabili globali all'interno del main.

2.3 random

La libreria random contiene le funzioni che servono per generare valori casuali utilizzate dai thread principali.

3 Ulteriori Dettagli Implementativi

3.1 Gestione dei Segnali

Per gestire i segnali SIGQUIT e SIGHUP sono stati utilizzati i seguenti flag globali:

```
volatile sig_atomic_t segnalesq; //SIGQUIT
volatile sig_atomic_t segnalesh; //SIGHUP
```

dove volatile sig_atomic_t è un tipo intero a cui si può accedere atomicamente e che quindi si può usare all'interno di un signal-handler. Ogni altro accesso a variabile globale (non dichiarata volatile sig_atomic_t) può avere un comportamento non definito. Inizialmente vengono settati a 0 nel main, ma nel momento in cui viene generato un segnale, tramite la propria funzione signal-handler impostano la propria variabile a 1. Immediatamente prima di uscire dalla funzione si tenta di risvegliare eventuali cassieri bloccati nelle variabili di condizione *cond[i]* per evitare che si verifichino deadlock.

3.2 Conteggio clienti attivi

Per rimanere informati sui clienti (thread) attivi nel supermercato si fa riferimento al seguente codice:

```
int nclienti;
pthread_mutex_t lockclienti=PTHREAD_MUTEX_INITIALIZER;
```

La variabile *nclienti* contiene il numero dei clienti attivi nel supermercato. Per mantenere aggiornata tale variabile essa viene incrementata all'ingresso di ogni cliente nel supermercato e viene decrementata dal cassiere dopo che questo è stato servito. A tal proposito poichè più thread modificano il valore di questa variabile è necessario introdurre una lock *lockclienti* in modo da poterla utilizzare in mutua esclusione. Parallelamente a questo codice sono state implementate tre funzioni in modo da poter eseguire le modifiche alla variabile *nclienti* in maniera pulita:

- void inc(int* nclienti); //incrementa *nclienti* in mutua esclusione.
- void dec(int* nclienti); //decrementa *nclienti* in mutua esclusione.
- int get(int* nclienti); //restituisce il valore di *nclienti*.

3.3 Scrittura statistiche su file

La scrittura delle statistiche riguardanti il Direttore, i Clienti e i Cassieri vengono eseguite sul file *filelog.txt*. Per poter eseguire tale scrittura è stato necessario utilizzare una variabile di mutua esclusione *filemtx*. Questo perché più thread clienti potrebbero scrivere sul file nello stesso istante.

4 Compilazione e Debugging

4.1 Compilazione

Per compilare il codice del programma è sufficiente scompattare l'archivio *lorenzo_angeli-corsoA.tar.gz*, spostarsi in tale directory ed eseguire il comando *make*. Vengono messi a disposizione altri comandi come *make test2* utile per eseguire un test del programma *supermercato* con invio di un segnale SIGHUP dopo 25s, il comando *make test1* simile al test2 ma con la differenza che questo invia un segnale SIGQUIT, il comando *make clean* per ripulire gli eventuali file creati in precedenza con l'esecuzione di *make* ed infine il comando *make all* che consente di eseguire il programma *supermercato*. Attenzione però perché in questo caso il programma non riceve nessun segnale SIGHUP/SIGQUIT in automatico ma è necessario inviare rispettivamente i comandi *killall -1 supermercato* o *killall -3 supermercato*.

4.2 Debugging

Per eseguire il debugging è sufficiente modificare i flag che si trovano nell'intestazione del file *supermercato.c* dal valore 0 al valore 1. Ogni flag fa riferimento a una parte del codice diversa che può essere debuggata.

```
#define DEBUGC 0 //DEBUG cliente
#define DEBUGCA 0 //DEBUG cassiere
#define DEBUGD 0 //DEBUG direttore
#define DEBUGM 0 //DEBUG main
```