Giuseppe Muschetta 564026 corso A, Prof. Prencipe

Il progetto contiene i seguenti file:

- analisi.sh
- cashier.c
- cashier.h
- config1.txt
- config2.txt
- customer.c
- customer.h
- director.c (server)
- director_core.c
- director_core.h
- macro.h
- Makefile
- market_core.c
- market_core.h
- queue.c
- queue.h
- relazione.pdf
- shared.c
- shared.h
- supermarket.c (client)

Il file shared.c è condiviso dai processi direttore e supermercato, e contiene le funzioni utilizzate per la scrittura sul socket, ovvero writen, readn, write_int_to_socket e read_int_from_socket. Le prime due sono come le classiche system calls write e read, ma implementate in modo da riprendere la loro esecuzione in caso di interruzione. Le altre riutilizzano (wrappano) rispettivamente readn e writen per scrivere degli interi sul socket.

Il file shared.h contiene i prototipi delle funzioni implementate in shared.c, tutte le librerie che servono e delle macro con valori interi, utilizzate come messaggi prefabbricati per la comunicazione via socket tra direttore e supermercato per mezzo delle funzioni write_int_to_socket e read_int_from_socket citate poco prima.

L'header macro.h contiene tutte la macro utilizzate nel progetto escluse quelle relative ai messaggi, che si trovano in shared.h. Tra queste ci sono anche le macro utilizzate per la gestione degli errori.

Il file queue.h contiene la definizione della struct node_t e della struct queue_t, ovvero i nodi della coda e la coda stessa. La struct queue_t contiene al suo interno variabili puntatore di tipo node_t utilizzate per riferirsi al primo e all'ultimo nodo della coda. L'header queue.h contiene anche i prototipi delle funzioni implementate in queue.c: init_queue per inizializzare una coda, delete_queue per deallocarla, push per mettere un nodo in fondo alla coda, pop per togliere un nodo dalla cima della coda (implementazione FIFO), remove_from_the_middle per togliere un nodo da una posizione qualsiasi (utile nell'algoritmo decisionale, nel caso in cui un cliente dovesse decidere di uscire dalla sua coda), e infine nodes_to_head per contare il numero di nodi dal nodo passato come parametro al nodo in testa alla coda (utilizzato anche questo nell'algoritmo decisionale per permettere ad un cliente di contare quanti ne ha davanti in coda).

I file director_core.c e market_core.c sono rispettivamente i file che implementano tutte le funzioni utilizzate in director.c e supermarket.c, mentre director_core.h e market_core.h sono i relativi header.

All'avvio, i processi director (server) e supermarket (client) chiamano rispettivamente le funzioni gestoreSegnaliDirettore e gestoreSegnaliMarket. Entrambe mascherano i segnali SIGHUP, SIGQUIT e SIGALRM (utilizzata dal direttore per avvisare il supermercato che ha preso una decisione di apertura o chiusura di una cassa) e ignorano il segnale SIGPIPE. La funzione gestoreSegnaliDirettore crea inoltre un thread signal_handler incaricato di gestire i segnali. Questo thread si mette in attesa di uno dei segnali mascherati in precedenza: in caso di SIGHUP o SIGQUIT lo inoltra al supermercato, mentre in caso di SIGALRM lo inoltra alla funzione decision presente in director_core.c (vedi dopo). Il thread esce quando il supermercato viene chiuso, ovvero quando tutti i threads del processo supermarket sono usciti.

Successivamente i processi director e supermarket si scambiano il file di configurazione e i pid dei due processi, e leggono (parsano) il file di configurazione per memorizzare i parametri necessari. Il direttore legge il file con la funzione parsingDirettore, la quale memorizza i parametri in una variabile globale di tipo director_args_t presente in director_core.c. Analogamente in supermarket.c viene letto il file di configurazione con parsingMarket e vengono memorizzati i parametri in una variabile globale di tipo supermarket_args_t presente in market_core.c. Nell'header market_core.h è definita la struct supermarket_args_t, mentre nei files customer.h e cashier.h sono definite le struct cliente e cassa_t contenenti tutti i parametri relativi a clienti e casse.

Fatto ciò, il processo director resta in attesa di connessioni da parte dei thread del processo supermarket finché il supermercato non viene chiuso. Ogni volta che una connessione viene stabilita, viene creato un thread worker che ascolta e soddisfa la richiesta. Il thread worker viene creato con creaThreadWorker, e la funzione eseguita dal thread è seleziona_compito, la quale svolge un'azione diversa in base al tipo di richiesta. In particolare, nel caso la richiesta consista in un aggiornamento sul numero di clienti in coda da parte di un cassiere, alla fine dell'aggiornamento viene chiamata la funzione decision. Questa per prima cosa decide che cassa aprire o chiudere e manda un segnale SIGALRM al processo supermarket, per poi fermarsi su una pthread_cond_wait in attesa che supermarket mandi un SIGALRM al processo director per informarlo che è pronto a ricevere l'ordine. Il segnale SIGALRM arrivato al processo director viene catturato dal thread signal_handler, che con una pthread_cond_signal risveglia il thread worker in decision dalla wait, che a quel punto deve solo informare il processo supermarket sulla decisione presa.

Come detto, il processo director smette di attendere connessioni da parte del processo supermarket quando quest'ultimo è rimasto senza thread attivi. A quel punto attende che il processo supermarket termini per poi terminare pure lui.

Il processo supermarket, una volta letto il file di configurazione, chiama la funzione aperturaSupermercato. Per prima cosa viene allocato un array di tipo cassa_t chiamato casse con la dimensione letta nel file di configurazione (il numero massimo di casse) e una coda di tipo queue_t chiamata coda_clienti_usciti, dove vengono accodati tutti i dati relativi ai clienti usciti dal supermercato (variabili di tipo cliente_t). Dopo aver inizializzato tutti i parametri necessari, crea i thread cassieri, i thread clienti e un thread entry_handler che si occupa di far entrare i clienti una volta uscita la soglia descritta nel file di configurazione.

Fatto ciò, il processo supermarket si ferma in attesa di segnali SIGQUIT, SIGHUP e SIGALRM, mentre i thread fanno il loro lavoro. In caso di SIGALRM significa che il direttore vuole aprire o chiudere una cassa, e quindi crea un threadOperaio che ascolta la richiesta del director. Questo thread per prima cosa manda SIGALRM al processo director (il segnale viene catturato dal thread signal_handler, il quale risveglia il thread worker dalla pthread_cond_signal nella funzione decision), e poi riceve l'ordine. Nel caso sia un ordine di apertura allora apre la cassa scelta dal direttore, altrimenti notifica alla cassa che deve chiudere. Dopo un segnale SIGALRM il processo supermarket si rimette immediatamente in attesa di altri segnali.

In caso di SIGQUIT il processo supermarket esegue la funzione evacuazioneDiEmergenza che si occuperà di chiudere il supermercato senza attendere che i clienti abbiano terminato gli acquisti. In caso di SIGHUP, infine, esegue la funzione chiusuraSupermercato che si occuperà di chiudere il supermercato attendendo che i clienti abbiano terminato gli acquisti. Se viene ricevuto un segnale SIGQUIT o SIGALRM, il processo supermarket non si rimette in attesa di segnali, e prima di terminare chiama la funzione scriviLogfile per la creazione del file di log.

Le funzioni evacuazioneDiEmergenza e chiusuraSupermercato fanno uso di una variabile globale booleana presente in market_core.c chiamata accesso_consentito, che indica se è consentito l'ingresso ai clienti nel supermercato. Questa variabile viene utilizzata per uscire da alcuni cicli while nei thread cassieri e nel thread entry_handler, e quindi dai thread stessi. La funzione evacuazioneDiEmergenza inoltre utilizza un'altra variabile globale booleana chiamata evacuation, anch'essa presente in market_core.c. Anche questa viene utilizzata per uscire dai cicli nei thread, ma a differenza di accesso_consentito viene utilizzata anche nei thread clienti per farli uscire senza aspettare di essere serviti.

All'interno di market_core.c si trovano tutti i thread del processo supermarket (clienti, cassieri, entry_handler e il threadOperaio). Ai thread clienti che vengono creati viene passata una variabile di tipo cliente_t come argomento, che contiene tutti i parametri necessari al cliente e quelli che dovranno essere settati per la creazione del file di log.

Un cliente che entra nel supermercato deve scegliere gli articoli e mettersi in coda. Successivamente, dopo ogni intervallo di tempo la cui ampiezza è descritta nel file di configurazione, decide se cambiare cassa. Ogni volta che un cliente si mette in coda ad una cassa esegue una pthread_cond_signal sulla condition variable della cassa. Infatti, nella struct cassa_t è presente una variabile di tipo pthread_cond_t, in modo che ogni cassa abbia la sua variabile di condizione: quando una cassa ha 0 clienti in coda, attende con una pthread_cond_wait di essere risvegliata. A risvegliare un thread cassiere dalla pthread_cond_wait può essere anche il threadOperaio, nel caso arrivasse l'ordine dal direttore di chiudere quella cassa.

Altra particolarità dei thread cassieri è che al momento della creazione creano un thread aggiorna_direttore incaricato, per l'appunto, di aggiornare il processo director sul numero di clienti in coda.

Infine, ogni volta che un cliente esce dal supermercato, esegue una pthread_cond_signal per risvegliare il thread entry_handler, il quale resta in attesa con una pthread_cond_wait che sia uscita la soglia di clienti indicata nel file di configurazione.