Progetto di Laboratorio di Sistemi Operativi

Anno accademico - 2020/2021

Pinna Matteo

UNIVERSITÀ DI PISA

Indice

1. Introduzione
2. Headers
   2.1 cashier.h
   2.2 customer.h
   2.3 utils.h
   2.4 myqueue.h
   2.5 mylist.h
   2.6 mylist2.h
3. Supermarket
   3.1 th_doorman
   3.2 th_spawner
   3.3 th_cashiers
   3.4 th_handler
   3.5 th_customer
4. Manager
   4.1 th_listener
   4.2 th_signal
5. Socket
6. Istruzioni compilazione

1. Introduzione

Il progetto consiste in 2 file principali

- *manager.c*
- *supermarket.c*

e in 6 header files contenuti nella directory *include*

- *cashier.h*
- *customer.h*
- *utils.h*
- *myqueue.h*
- *mylist.h*
- *mylist2.h*

2. Headers

I vari file headers utilizzati nel progetto contengono sia la definizione che l'implementazione delle varie funzioni, non necessitando così di un corrispondente file .c con successivo linking ad una libreria. Per evitare errori a causa di multiple dichiarazioni, essendo alcuni headers inclusi in più parti del progetto, sono state utilizzate le direttive *ifndef – define – endif* all'interno di tutti i sopracitati.

2.1. cashier.h

File header utilizzato tramite include in *supermarket.c*. Definisce la *struct cashierArgs_h* contenente le varie informazioni relative al singolo cassiere.

2.2. customer.h

File header utilizzato tramite include in *supermarket.c* e *myqueue.h*. Definisce la *struct customerArgs_h* contenente le varie informazioni relative al singolo utente. In particolare, tre valori vengono settati a 1 dal cassiere quando risulta necessario comunicare un determinato evento:

- **closing:** cassiere in chiusura
- **checkout:** il cliente sta venendo servito
- **paid:** il cliente è stato servito e può uscire

2.3. utils.h

File header contenente alcune macro, struct e varie funzioni di utilità per una maggiore leggibilità del codice all'interno di *supermarket.c e manager.c*:

- **notification_t:** struct utilizzata per la ricezione dei messaggi
- **SYSCALL:** macro per la gestione di errori di chiamate di sistema
- **write_h:** funzione per effettuare scrittura su socket
- **read_h:** funzione per effettuare lettura da socket
- **unlink:** funzione per la pulizia delle socket

Le restanti *MUTEX_LOCK*, *MUTEX_UNLOCK*, *COND_WAIT* e *TH_JOIN* si occupano, come si può immaginare dal nome, dell'esecuzione e controllo errori rispettivamente di *pthread_mutex_lock*, *pthread_mutex_unlock*, *pthread_cond_wait* e *pthread_join*.

2.4. myqueue.h
Header file utilizzato tramite include in *supermarket.c*. Definisce una coda di tipo FIFO utilizzata per la rappresentazione delle code alle casse, gli elementi all'interno di quest'ultima sono rappresentati da un elemento *customerArgs_h*, il quale permette al cassiere, nel momento in cui preleva il primo cliente in coda tramite la funzione *takeCustomer*, di accedere ai dati del cliente che sta servendo e impostare determinati flag precedentemente descritti.

2.5. mylist.h
Header file utilizzato tramite include in *cashier.h*. Definisce una linkedlist utilizzata per memorizzare i tempi di ogni singolo cliente servito e di ogni apertura – chiusura di un cassiere. Questo obiettivo viene raggiunto tramite la definizione di due elementi *mylist_h* all'interno della struct che contiene le informazioni della cassa.

2.6. mylist2.h
Header file utilizzato tramite include in *supermarket.c*. Definisce una seconda linkedlist che viene invece utilizzata per memorizzare tutti i dati dei clienti che entrano nel supermercato per la successiva trascrizione nel file di log. Per questo scopo viene definita la struct *customerAnalytics_h* all'interno dell'header file in questione, la quale contiene esclusivamente i dati a cui siamo interessati per la trascrizione.

3. Supermarket
Il file principale *supermarket.c* contiene i seguenti thread:

3.1 th_doorman
Si occupa di gestire l'ingresso dei clienti nel supermercato. Al momento dell'apertura genera un numero predefinito di thread cliente in modalità detached mentre per il continuo ricambio di clienti attende che il numero all'interno del supermercato scenda al di sotto di una soglia sempre predefinita.

3.2 th_spawner
Si occupa di gestire apertura e chiusura delle casse, escluse quelle inizializzate dal *main* al momento dell'apertura, in seguito alla ricezione di opportune notifiche da parte del manager. Queste operazioni vengono portate a termine tramite l'utilizzo di due flag *addCashier* e *closeCashier*, i quali vengono settati a 1 dai thread che gestiscono la comunicazione con il manager.

3.3 th_cashiers
Array di threads che rappresentano i vari cassieri presenti all'interno del supermercato. I singoli thread cassa comunicano determinati eventi ai vari clienti tramite il settaggio a 1 di tre flag, contenuti nelle loro struct, che abbiamo precedentemente descritto.

3.4 th_handler
Ogni thread cassa contiene al suo interno uno specifico thread che si occupa della gestione della comunicazione con il manager. L'implementazione di un thread che si occupi esclusivamente della comunicazione cassa-manager è stata effettuata per evitare che il thread cassa ritardasse il servizio dei propri clienti. Con lo stesso obiettivo l'handler è stato implementato in modo tale che acquisisca la lock sulla coda solo nel momento della lettura della lunghezza di quest'ultima, se l'avesse mantenuta per tutta la durata dell'interazione con il manager avremmo si avuto dati consistenti, nel senso che al momento della ricezione di una notifica di aggiunta o rimozione cassa la situazione della coda sarebbe ancora stata la stessa al momento dell'invio, tuttavia si sarebbe creato un ulteriore ritardo nel servizio offerto dal cassiere. Quindi, per impedire la chiusura o apertura di una cassa quando le condizioni non sono più rispettate, ogni handler verifica, nel momento in cui riceve una risposta dal manager, che la situazione della coda sia ancora tale da necessitare l'attuazione dell'operazione richiesta.

3.5 th_customer
Per la realizzazione dell'algoritmo di cambio coda dei clienti si è optato per una implementazione relativamente semplice: il tentativo di cambio coda avviene, banalmente, solo se esiste almeno un'altra cassa

aperta e si ritiene concluso nel momento in cui il cliente analizza la coda di esattamente una cassa tramite il confronto tra il numero di clienti davanti a se e la lunghezza della coda in esame.

Nel *main* del supermercato avvengono tutte le operazioni di I/O su file, quindi lettura dei valori di default dal file config e salvataggio dei dati di analisi sul file di log, e la ricezione dei segnali di SIGHUP e SIGQUIT da parte del manager. La gestione di questi ultimi avviene impostando una variabile globale, rispettivamente *closing* per SIGHUP, da non confondere con il *closing* contenuto nelle struct dei clienti, e *quit* per SIGQUIT.

4. Manager

Il file principale *manager.c* contiene i seguenti thread:

4.1 th_listener

Si occupa di rimanere in ascolto di eventuali notifiche da parte di clienti e cassieri. Al momento della ricezione di una richiesta viene generato un thread specifico *th*, lanciato in modalità detached, che si occuperà dell'effettiva gestione della richiesta. In particolare, nel caso di notifica da parte di un cliente, la connessione sarà immediatamente chiusa una volta terminata la sua gestione, mentre per quanto riguarda le notifiche dei cassieri, il canale verrà mantenuto per successive comunicazioni fino al momento della chiusura della cassa stessa. In questo modo non risulta necessario creare una nuova connessione ogni volta che riceviamo una notifica dalla medesima cassa.

4.2 th_signal

Comunica al supermercato di effettuare una chiusura immediata o graduale nel momento in cui il manager riceve un segnale di SIGHUP o SIGQUIT rispettivamente. Per assicurare che la terminazione del manager sia successiva al supermercato, il *th_signal* attende una risposta di conferma di avvenuta chiusura da quest'ultimo prima di terminare.

5. Socket

La comunicazione tra i due programmi principali avviene tramite l'utilizzo di due socket: *toM_socket* e *toSM_socket*. La prima viene utilizzata lato *supermarket.c* per l'invio delle notifiche, la cui gestione avviene tramite l'implementazione di *manager.c* come fosse un Server multithread, tenuto conto del fatto che viene generato un thread per ogni richiesta di connessione. La seconda è invece utilizzata lato *manager.c* per l'invio di segnali a *supermarket.c*.

6. Istruzioni compilazione

Per la compilazione del progetto è sufficiente, come richiesto nelle specifiche, decomprimerlo in una cartella vuota ed eseguire il comando *make* del Makefile che genererà i file di output di *supermarket.c* e *manager.c*.
Per l'esecuzione dei due casi di test i comandi sono rispettivamente *make test1* e *make test2*, alla loro conclusione tutti i file temporanei vengono eliminati.
Una volta conclusa la sessione di testing del progetto è possibile pulire i file di output *supermarket.o* e *manager.o* tramite il comando *make clean*.