UNIVERSITÀ DI PISA

Corso di Laurea Triennale in Informatica

A.A. 2019-2020

Sistemi Operativi e Laboratorio

Corso A

II Semestre

Relazione progetto di corso

Studente

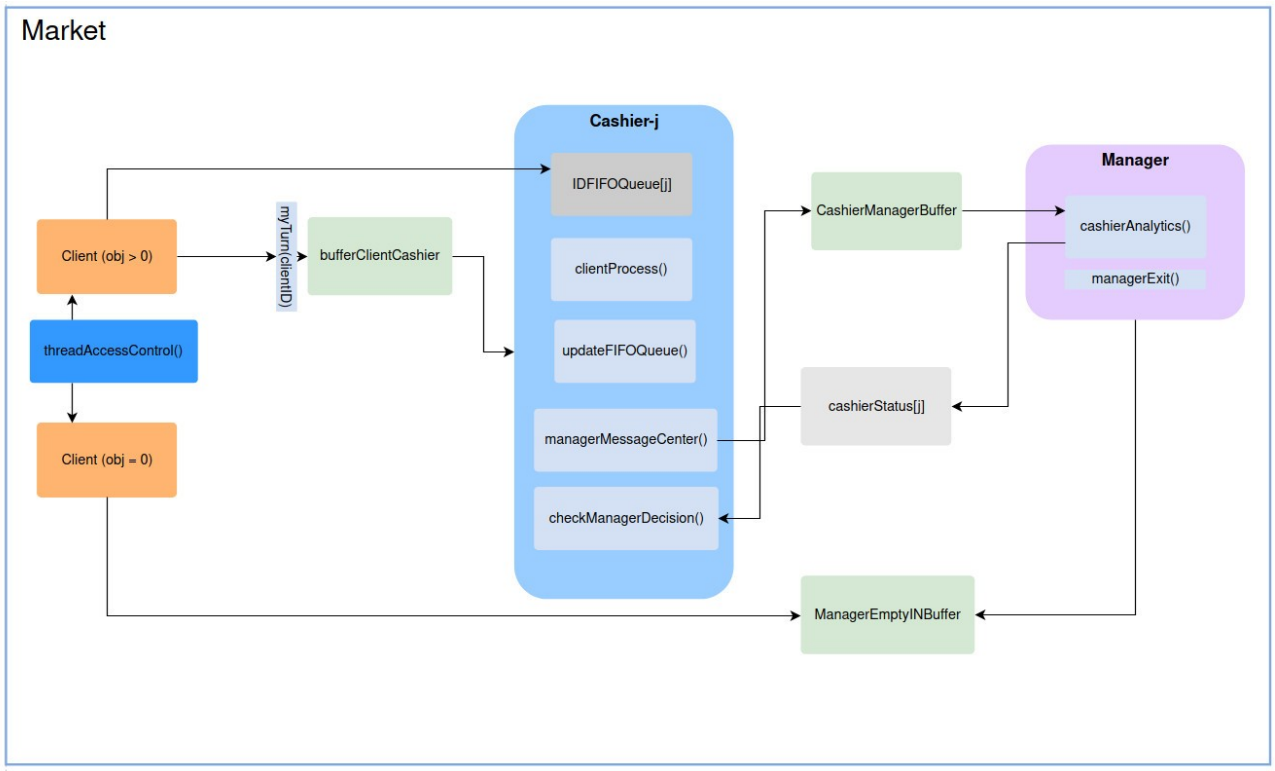
Luca Giovambattista Pinta
Matricola: 579458

Docenti

Prof. G. Prencipe
Prof. M. A. Bonuccelli

Progettazione

La fase di progettazione è stata guidata dalla visione della specifica e nella scelta di segmentare, isolando ove possibile, le funzionalità ed i meccanismi interni di ogni singola componente da i punti di comunicazioni con le altre componenti. Al fine di mantenere efficiente i meccanismi interni, è stato scelto di istanziare le componenti sotto identificate con relativi thread principali e sotto-thread di cui si avvalgono per espletare funzioni di gestione e comunicazione interne quanto esterne.
Al fine di implementare la sincronizzazione e la comunicazione si è fatto ampio utilizzo di variabili condizione associate alle mutex per l'accesso a buffer o code. Per ogni struttura dati condivisa tra le 4 componenti si utilizza una lock e ove necessiti un'attesa passiva si fa uso di condition variables. Nello specifico:



Market (Processo): poiché punto di ingresso del programma si occupa di allocare e inizializzare tutte le strutture dati e le componenti utilizzate. Il market leggera dal file di configurazione "config.txt" il valore dei parametri di esecuzione ponendoli all'interno di una variabile di tipo **config** che oltre ai parametri noti contiene un array contenente le tempistiche dei singoli e due flag per i segnali di SIGQUIT e SIGHUP (dichiarati volatile sig_atomic_t). Al fine di gestire efficacemente il controllo di soglia C-E l'ingresso di nuovi clienti è mediato dal ***threadAccessControl()***. Per delega del thread manager si occupa inoltre dell'istanziazione dei cassieri in caso di apertura tramite la funzione *cashierLauncher()* e dell'istanziazione di nuovi clienti tramite la funzione

clientLauncher(). Gestisce il setting dei segnali si SIGHUP e SIGQUIT e al termine dell' esecuzione gestisce la creazione del file di log.

Client (thread): in maniera sequenziale i client si iscrivono nella coda FIFO del cassiere selezionato e rimarranno in attesa sulla coda ***IDFIFOQueue*** stessa finché non saranno in testa alla coda. Risvegliati dal cassiere, accederanno al buffer, posizionando il messaggio e usciranno dal market. Nel caso di un cliente con zero prodotti acquistati i clienti saranno in attesa sul buffer ***ManagerEmptyINBuffe*r**.

Cashier (thread): i cassieri possono trovarsi in 3 differenti stati indicati nell' array *cashierStatus:* **1** se cassa attiva, -**1** se cassa già chiusa (dunque relativo thread ucciso) e **0** per cassa in chiusura (stato di transizione indotto dalla decisione del manager di chiudere una determinata cassa). I cassieri attenderanno sul buffer *bufferClientCashier* finché vuoto, verranno risvegliati dal cliente e dunque verrà invocata *processClient()* per il cliente corrente, aggiornata la coda FIFO tramite *updateFIFOQueue()* per eventuali nuovi clienti e per comunicare i dati aggiornati della cassa tramite *managerMessageCenter()* che fa uso del buffer condiviso ***CashierManagerBuffer*** ed infine si effettuerà il controllo di eventuali ordini di chiusura da parte del manager tramite *checkManagerDecision()* che effettua un controllo su *cashierStatus[cashierID]* che se settato a 0 dal manager induce il cassiere a servire l' ultimo cliente ed infine uscire.

Manager (thread): il thread manager si decompone in un thread ausiliario *managerDecision()* che tramite la funzione interna *cashierAnalytics()* si occupa della gestione dei dati delle casse (tramite ***CashierManagerBuffer***) e dei relativi ordini di apertura/chiusura casse.
Il core del thread manager si occupa invece della gestione dei clienti con zero prodotti effettuando attesa passiva sul buffer condiviso ***ManagerEmptyINBuffer***: il meccanismo di autorizzazione all' uscita richiede di effettuare una richiesta di uscita sul buffer; tale richiesta verrà approvata dal manager e lasciata sul buffer finché il cliente associato alla richiesta non rimuoverà la richiesta e uscirà dal market.

Libreria
Seppur non strettamente necessario si è scelto di utilizzare una libreria statica contenente i 4 moduli del progetto incluso l' header file (lib/*marketlib.h*) al fine di una migliore suddivisione e organizzazione del codice e nella correzione degli errori.

Segnali
I segnali vengono registrati nel modulo market ed i rispettivi handler si limitano porre le due variabili flag (*volatile sig_atomic_t*) contenute nello struct di

configurazione al valore 1. All' arrivo del segnale di **SIGQUIT** il manager provvederà a risvegliare eventuali cassieri in attesa sul rispettiov buffer ed eventuali client in attesa sul buffer ***ManagerEmptyINBuffer.*** I client usciranno autonomamente così come i cassieri. Il manager chiuderà il sotto-thread *decisionManager()* e conseguentemente uscirà tramite una pthread_exit() tornano al processo market che aveva effettuato una pthread_join.
Analogamente per il segnale di SIGHUP il manager interromperà qualsiasi attività di apertura/chiusura delle casse e il thread ***threadAccessControl()*** impedirà l' accesso di nuovi clienti al market. Il manager risveglierà eventuali cassieri in attesa che in caso di coda FIFO vuota usciranno. I rimanenti cassieri processeranno i clienti rimasti in coda prima di terminare anch' essi.

Configurazione
Il file di configurazione si compone dei parametri indicati nel testo del progetto e per scelta implementativa il tempo di attesa di ogni prodotto è indicato da Nj dove j indica il prodotto j-esimo seguito dal tempo di attesa di quel singolo prodotto.
E' tuttavia possibile specificare solo il numero massimo prodotti: in questo caso i relativi tempi di attesa dei prodotti se non presenti nel file saranno settati a 1.

Make File
Per eseguire il progetto sarà necessario effettuare in ordine i comandi *make* e *make test.* Al fine di eliminare eventuali file residui (data.txt per il file di log o file di libreria libmarket.a) sarà necessario effettuare prima *make clean*.