UNIVERSITÀ DI PISA

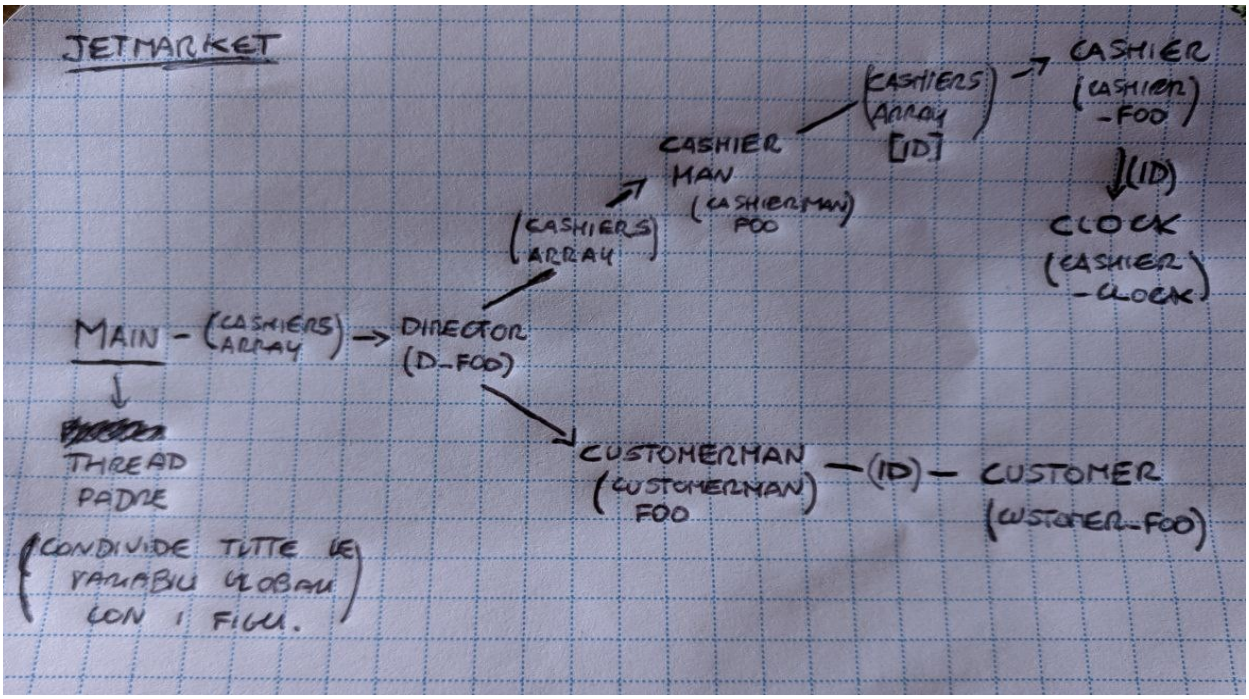
Relazione relativa al progetto di SOL 19/20

Insegnanti:
Prof. Giuseppe Prencipe
Prof. Maurizio Angelo Bonuccelli

Studente:
Francesco Giuseppe Ziccolella
588922 (Corso A)

Progettazione e strutture utilizzate:

Per la progettazione di questo caso sono partito dalla visione gerarchica dei thread e dal passaggio di argomenti , disegnando uno schema



Seguendo le specifiche del progetto ridotto il direttore è considerato un thread appartenente al processo market , quindi sfruttando la proprietà della condivisione dello spazio tra i vari thread ho introdotto una serie di variabili globali con variabili di supporto per la mutua esclusione :

CASSE:

queue ** qs= è un array di code che serve per simulare la struttura di un vero sistema di cassieri di un supermercato.

Ad affiancare questa struttura per gestire la sincronizzazione ho introdotto:

- **q_mutex** = usata quando un thread deve controllare se la cassa è aperta o per la misurarne la coda. (bloccando l'utilizzo delle code ai restanti)

- **q_cond** = usata per sincronizzare cassiere e clienti in caso di pagamento o chiusura cassa.

- **wake_cond** = usata per svegliare il thread cassiere all'entrata del primo cliente, alla chiamata del sigquit e alla chiusura del supermercato.

SISTEMA APERTURA/CHIUSURA CASSE:

closing_bit = è una Bitmap che indica se una cassa è in chiusura o no.
(0=non in chiusura, 1=in chiusura)

Ad affiancare questa struttura per gestire la concorrenza ho introdotto:

closing_bit_mutex : usata per sincronizzare il clock del cassiere con il direttore, per esempio, attraverso la closing_bit il direttore cerca le casse valide da aprire o chiudere.
La lettura del clock avviene ogni t_update millisec e la decisione avviene quando tutte le casse aggiornano il loro stato(lunghezza / apertura-chiusura)

SISTEMA DI USCITA

queue * exitqueue = una volta che il cliente ha pagato viene inserito nella exitqueue, il CustomerMan dovrà dargli il permesso di uscire.

Ad affiancare questa struttura per gestire la sincronizzazione ho introdotto:

exitqueque_mutex = usata quando un thread deve controllare la lunghezza di exitqueue , aggiungere o rimuovere un cliente. (bloccando l'utilizzo della coda ai restanti)

okcond = usata quando CustomerMan accetta l'uscita del cliente dal supermercato , quindi sveglia attraverso signal il cliente , il quale termina e scrive sul file di log le info.

CustomerNeedsToExit = usata per svegliare CustomerMan quando c'è almeno un cliente nella exitqueue.

SISTEMA AGGIORNAMENTO:

int * q_update = è una Bitmap che indica se la lunghezza di una cassa è stata aggiornata o no .
(la uso per dire al CashierMan che il clock ha aggiornato la qlenght)

Ad affiancare questa struttura per gestire la sincronizzazione ho introdotto:

pthread_mutex_t * update = usata per gestire la mutua esclusione su q_update[id] dove è il numero del cassiere

updated_mutex = Gestisce la concorrenza su tutto l'array updated per contare le casse apribili e chiudibili.

qlengh_updated_cond = Sveglia il CashierMan quando è stata aggiornata la qlenght

LIBRERIE EXTRA:
Per misurare il tempo e simulare il tempo di acquisto ho usato clock_gettime e nanosleep appartenenti alla libreria <time_h>.

//

Nel caso di sighup:

CustomerMan non fa entrare piu' clienti e fa uscire quelli rimanenti , nel frattempo I cassieri serviranno gli ultimi rimanenti nel supermercato.

Appena I clienti nel supermercato sono 0 , CustomerMan setta exit_broadcast=1 e sveglia i cassieri che con la condizione exit_broadcast=1 joinano il clock.

Nel frattempo il clock del cassiere avrà aspettato un intervallo di t_update e poichè sighup=1 allora terminerà.

CashierMan joina I customer thread ma essendo terminati, termina.(cassiere,clock e cashierman terminati).

CustomerMan joina I customer thread ma essendo terminati, termina.

CustomerMan e CashierMan (subthread di Director) terminati , termina anche d_foo.

//

HEADERS :

- Cashier_Functions.h : contiene la struct del cassiere e il metodo che la inizializza.

- Queue_Functions.h : contiene la struct del customer ,il metodo che la inizializza e le funzioni base della gestione di una coda.

- Cashier_Functions.h : contiene la struct del config , variabile globale , il metodo che effettua il parsing e un metodo che controlla che i valori siano accettabili.

MAKEFILE: (comandi)

test: - elimina I file creati da un'esecuzione vecchia se trova un vecchio .PID
- se non trova il file oggetto compila
- esegue il programma con I parametri del file passato come argomento (text2.txt)
- manda un segnale SIGHUP dopo 25 secondi
- esegue lo script e stampa a video i risultati

compile : compila il file oggetto

clean : elimina I file creati da un'esecuzione vecchia