# PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI 2019-2020

## PREFAZIONE

Progetto del corso di Laboratorio di Sistemi Operativi dell'Universita di Pisa, anno accademico 2019/2020.

Tutto il codice è stato scritto da me, ad accezione del parser Ini che è di proprietà di benhoyt (Link al git). Link al testo del progetto: Progetto.pdf

## OVERVIEW

Il progetto consiste nella realizzazione di un processo che simula la gestione di un supermercato. Le entità presenti in questo processo sono le seguenti:

1. **Direttore:** Ha il compito di aprire e chiudere le casse, gestire l'afflusso e l'uscita di clienti all'interno del supermercato e di monitoraggio globale.
2. **Casse:** Il loro ciclo di vita è quello di elaborare le richieste dei clienti, e ogni T millisecondi aggiornare il direttore del proprio stato interno.
3. **Cliente:** Una volta entrato nel supermercato raccoglie un numero random di prodotti e poi va alla cassa per pagare. Una volta processato dalla cassa chiede al direttore la possibilità di uscire.

## SCELTE DI IMPLEMENTAZIONE.

Il processo è strutturato come segue:

```
Processo ./bin/supermercato.o
└──Thread mainDirettore
    ├──Thread direttoreButtaDentro
    │   └──Thread MainCliente
    └──Thread DirettoreApriChiudi
        └──Thread mainCassa
            └──Thread updateDirettore
```

### DIRETTORE

Il thread principale è quello del Direttore, a sua volta genera due sotto-thread di gestione. Il primo, il ***direttoreButtaDentro*** si occupa di generare i thread Clienti e di controllarne l'afflusso. Per prima cosa genera C thread clienti, e poi controlla che il numero di clienti attivi nel supermercato non sia mai inferiore al valore soglia. In oltre gestisce le richieste di uscita da parte dei clienti. Il secondo, il ***direttoreApriChiudi***, per prima cosa inizializza un numero fissato di thread Cassa iniziali (fissato nel file di configurazione), e poi ne gestisce l'apertura/chiusura in base alle informazioni che le casse gli notificano ogni T ms. Se ad un ciclo apre una cassa al successivo ne chiude una. Non è detto che ad ogni iterazione apra o chiuda una cassa, dipende se tutte le casse aperte in quel momento gli hanno inviato un update e se almeno uno dei due check (valori soglia S1 e S2 del file di configurazione) sono rispettati.

## CASSA

Il thread viene creato dal ***direttoreButtaDentro*** e gli viene passata una struct ***infoCassa**** come argomento. Questa struct rappresenta le informazioni della cassa per tutta la durata del processo, non solo per la vita del thread. Una volta creato dal direttore cicla finchè non viene chiuso. Come prima cosa genera un thread di supporto che invia i propri dati al direttore, con scadenza regolare. Aspetta di avere dei clienti in coda e poi li processa, salvandosi i dati necessari in una struct globale. Un array di struct ***infoCassa**** viene generato all'inizio del processo, e contiene i dati globali di tutte le casse. In base alle politiche di apertura/chiusura del direttore può capitare che la stessa struttura dati venga aggiornata da più thread nel corso della vita del processo. Se un thread viene chiuso ad un tempo *t*, la struttura dati che lo identifica rimane presente come variabile globale all'interno dell'array sopracitato, e può essere passata come argomento ad un secondo thread più avanti.

## CLIENTE

Una volta generato, ha come argomento solo un ID che lo identifica. Genera un numero di prodotti Random (max P prodotti) e simula il loro acquisto con una *nanosleep* di *P_prodotti*T_cliente*. Successivamente entra in un ciclo il cui scopo è quello di scegliere una cassa da cui farsi processare. Entra in coda, e finchè non riceve una risposta dalla cassa rimane in attesa passiva. Se la cassa lo processa allora esce dal ciclo. Se la cassa viene chiusa nel frattempo allora esegue nuovamente una scelta tra le casse attualmente in coda. Se il cliente non ha prodotti, salta questa fase. Una volta processato aspetta l'approvazione dal direttore per poter uscire, e similmente alla cassa entra nella coda del Direttore.

## SCELTE DI IMPLEMENTAZIONE

Ogni Cassa è identificata da una struttura dati contenente le sue informazioni interne, tra cui il proprio ***ID***. Questo ID è fondamentale perché identifica la sua posizione all'interno dell'array ***InfoCasse*** e la propria coda dentro l'array ***CodaClienti***. Anche il direttore possiede una coda, ***CodaDirettore***.

Per processare i clienti da parte delle casse e del direttore ho usato una coda FIFO, implementata semplicemente come una linked-list con una push e una pop.

L'update da parte delle casse al direttore viene fatto tramite un thread di supporto, che setta un valore booleano all'interno di un array (per ogni cassa un indice, relativo al proprio ID) ogni qualvolta che notifica le proprie informazioni al direttore. Il direttore, da parte sua, aspetta che almeno una cassa esegua la notifica. Quando tutte le casse aperte in quel momento hanno effettuato una notifica allora sceglie l'indice di cassa da aprire/chiudere, in base ad un valore ***aprichiudi*** (1 o 0) e se il valore soglia viene superato (S1 se devo chiudere, S2 se devo aprire).

Tutte le operazioni all'interno dei thread tengono conto di una possibile interruzione relativa ai segnali sigHUP e sigQUIT, nel caso in cui avvengano le operazioni continuano fino al compimento dell'obbiettivo del thread (sigHUP) o fino ad uno stato neutro (sigQUIT).

Per ultimo, ho eseguito il parsing del file di configurazione con una piccola libreria esterna (licenza MIT, al momento della consegna senza copiright) di proprietà di benhoyt (Link al git). Ho scelto questa piccola libreria perché avevo già manualità con il suo utilizzo avendola già usata in passato per progetti personali e per l'estrema flessibilità dei file .ini. Non essendoci vincoli sull'estensione del file di configurazione e il suo parsing il fulcro del progetto ho ritenuto che fosse un buon tool da usare.

## ESEGUIRE IL PROGETTO

All’interno dell’archivio è possibile trovare il file Makefile con cui compilare il progetto.

Con il comando ***make clean*** si pulisce la directory dei vari codici oggetto, file di log e file storage per il pid.

Con il comando ***make test*** si esegue il progetto, e dopo 25 secondi viene inviato un segnale di tipi SIGHUP per confermare la chiusura del supermercato. Una volta terminato il processo si esegue lo script di analisi che recupera tutte le informazioni dal file di log.

Con il comando ***make test2*** si esegue il progetto come in precedenza, ma viene inviato un segnale SIGQUIT.