# Progetto di laboratorio

# Sistemi Operativi

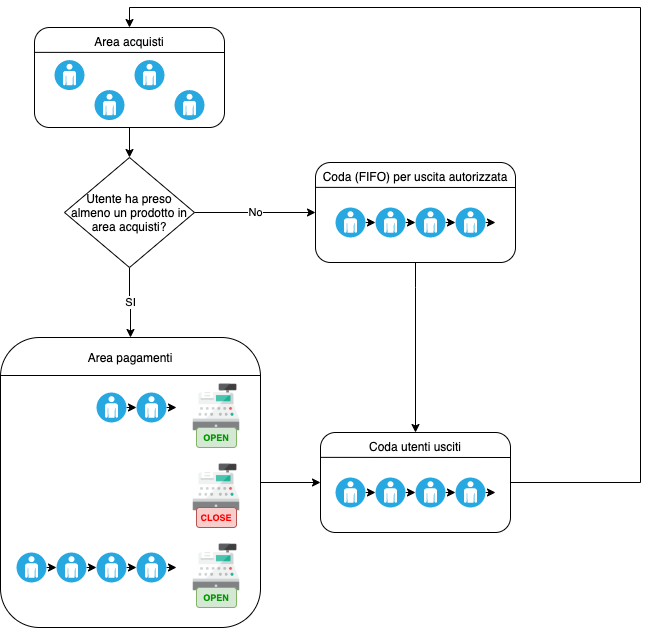
# A.A.2019/2020 - Corso A

Andrea Del Corto

matricola: 561446

# Scelte progettuali

La prima fase del progetto è stata quelle di definire chiaramente le entità coinvolte nel progetto. A tale scopo è stato realizzato il seguente schema che da una visione del supermercato incentrata in particolare sullo spostamento degli utenti all’interno del supermercato durante la simulazione:



Successivamente è stato definito l’insieme dei thread e i loro rispettivi compiti:

**Market thread:**

Il programma ha un unico thread di questo tipo che gestisce i seguenti aspetti della simulazione:

1. Inizializza tutte le strutture dati necessarie per avviare la simulazione di un supermercato.
2. Quando in “Coda usciti” arrivano E utenti, si occupa di resettare i loro attributi per poi inserirli nuovamente in “Area acquisti”. Questo riutilizzo delle strutture dati permette di ridurre sensibilmente il numero di chiamate malloc e di creazioni di thread, con un conseguente riduzione del consumo delle risorse computazionali.
3. Quando un segnale di chiusura (SIGHUP o SIGQUIT) viene ricevuto, questo thread si occupa di gestire la chiusura facendo terminare in modo controllato tutti gli altri thread.

**User thread:**

Il programma ha C thread di questo tipo e ognuno di essi gestisce i seguenti aspetti della simulazione:

1. Gestisce una struttura dati “User” che contiene le informazioni relative ad un utente.
2. Registra timestamp di ingresso nell’area di shopping.
3. Attende un periodo di tempo, definito in modo casuale compreso fra 10 e T millisecondi per simulare tempo di shopping.
4. La struttura dati “User” gestita dal thread viene spostata in una cassa aperta di “Area pagamenti” scelta casualmente se è stato preso almeno un prodotto, altrimenti viene spostata in “Coda (FIFO) per uscita autorizzata”.

**CashDesk service thread:**

Il programma ha K thread di questo tipo e ognuno di essi gestisce i seguenti aspetti della simulazione:

1. Gestisce una struttura dati “CashDesk” che contiene le informazioni relative ad una cassa.
2. Monitora i cambi di stato della cassa (aperta/chiusa) per tenere traccia del tempo totale di apertura e del numero di chiusure.
3. Ogni utente che viene inserito nella coda della cassa viene servito simulando un’attesa di servizio pari al seguente tempo:

|  |
| --- |
| T(n) = c1 + n\*NP    n: numero di prodotti presi dal ciente  c1: tempo costante diverso per ogni cassiere compreso fra 20-80 ms (viene determinato in modo casuale alla creazione della struttura dati CashDesk)  NP: è un parametro del file di configurazione che definisce il tempo in millisecondi necessario per processare un prodotto. |

1. Dopo essere stato servito l’utente viene spostato nella coda di uscita, in modo da permettere al Market thread di utilizzarlo per una nuova simulazione.

**CashDesk notification thread:**

Ogni “CashDesk service thread” istanzia un thread ausiliario dedicato all’invio delle notifiche al thread direttore (id della cassa, numero utenti in coda nella cassa, stato della cassa). Ogni TD millisecondi (TD è un parametro del file di configurazione), questo thread inserisce una apposita struttura dati contenente le informazioni di stato della cassa in una coda gestita dal thread direttore.

**Director desk handling thread:**

Il programma ha un unico thread di questo tipo e si occupa di gestire i seguenti aspetti della simulazione:

1. Gestisce le notifiche ricevute da i thread ”CashDesk notification thread”:

Questo thread ha un vettore di notifiche chiamato “lastReceivedMsg” di K elementi il cui scopo è quello di tenere traccia delle notifiche più recenti relative a ciascuna cassa. Inizialmente gli elementi di questo vettore hanno valore “NULL” e ogni volta che viene ricevuta una notifica essa viene messa nel vettore “lastReceivedMsg” nella posizione relativa alla cassa che ha generato la notifica. Quando il vettore non contiene più elementi uguali a NULL significa che tutte le casse hanno inviato almeno una notifica ed è quindi il momento di decidere se aprire o chiudere una cassa.

Questa decisione viene presa come descritto nel testo del progetto e la cassa da chiudere/aprire viene scelta in modo casuale.

**Director authorization queue thread:**

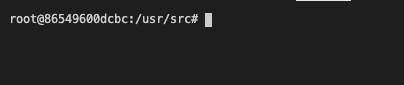
Ogni “Director desk handling thread” istanzia un thread ausiliario dedicato alla gestione della coda degli utenti in attesa di ricevere l’autorizzazione ad uscire. Questo thread si limita semplicemente ad spostare gli utenti nella coda di uscita, in modo da permettere al Market thread di utilizzarli per una nuova simulazione.

# Parti supplementari

# Docker

Il progetto è stato interamente sviluppato su una macchina virtuale con Ubuntu 16.04, ma il programma potrebbe essere valutato anche su Debian. Essendo entrambi sistemi UNIX non dovrebbero esserci problemi nel compilare ed eseguire il progetto su entrambi, ma per esserne più sicuri sono state fatte delle prove anche su Debian. Per poter effettuare le prove su Debian è stata utilizzata un’apposita immagine [Docker](https://docs.docker.com/get-started/overview/) derivata dall’ultima release stabile di Debian il cui nome è: market.

All’interno del progetto è stato definito un file chiamato: “Dockerfile” il cui scopo è quello di definire l’immagine market. Utilizzando il comando “make docker\_build” l’immagine verrà creata e potrà essere messa in esecuzione con il comando “make docker\_run”. Dopo aver eseguito il comando “make docker\_run” nel terminale dovrebbe presentarsi una situazione di questo tipo:



Questo significa che l’immagine market è stata messa in esecuzione correttamente, pertanto da questo momento in poi tutti i comandi che saranno digitati saranno eseguiti in ambiente Debian. Il terminale del container (immagine in esecuzione prende questo nome secondo la terminologia Docker) market di default viene posizionato nella cartella “usr/src”, la quale è un link che contiene tutti i file del progetto che si trovano sull’host che ha lanciato il container. Infatti eseguendo il comando ls si vedono i file del progetto presenti sull’host:



# Doxygen

[Doxygen](http://www.doxygen.org/index.html) è un tool che permette di generare automaticamente la documentazione di un progetto scritto in C++, C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL o in D (supportato solo in parte).

Per questo progetto i commenti presenti nei file sorgenti sono stati scritti seguendo lo standard Doxygen descritto sulla pagina ufficiale: <https://www.doxygen.nl/manual/starting.html#step3>.

La documentazione generata con Doxygen si trova nella seguente cartella: “doxyDocs”.