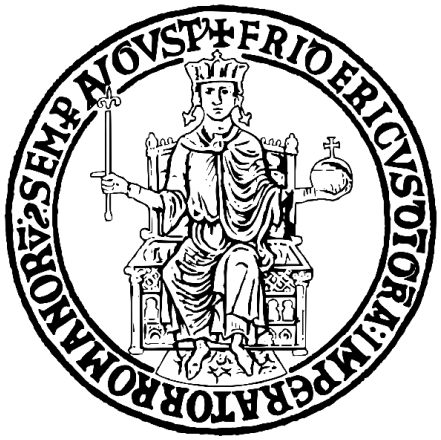
Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea in Informatica - Insegnamento di Laboratorio di Sistemi Operativi

Anno Accademico 2023/2024

Progettazione di "*Miao Market*"

Piattaforma per la Simulazione di un Supermercato

Antonio Abbatiello - N86003037

Leonardo Colamarino – N86003586

09/2024

Indice

SOMMARIO

CAPITOLO 1 – DESCRIZIONE DEL PROGETTO

1.1 INTRODUZIONE AL PROGETTO

Il progetto **“Miao Market”** si propone di creare una simulazione che riproduca il funzionamento di un supermercato, con casse e clienti gestiti in modo dinamico. Il sistema simula il flusso di persone all’interno del supermercato, regolando l’ingresso e l’uscita dei clienti per ottimizzare l'efficienza del processo di acquisto e pagamento.

La piattaforma è composta da un server che gestisce tutta la logica di funzionamento del supermercato e le interazioni con i client, i quali possono operare in due modalità: automatica, dove il comportamento del cliente è simulato, e interattiva, in cui l'utente può controllare direttamente il processo di acquisto. L'integrazione tra server e client è realizzata attraverso l'uso di socket, con protocolli di comunicazione affidabili che permettono una gestione fluida delle richieste.

1.2 CLIENT

Il client rappresenta un cliente che accede al supermercato per fare acquisti. Può essere eseguito in due modalità:

- **Automatica**: il comportamento del cliente è deciso dal sistema,
- **Interattiva**: il cliente è controllato direttamente dall'utente.

In modalità interattiva, una volta entrati nel supermercato, si presenta un'interfaccia grafica per la selezione dei prodotti da aggiungere al carrello. Se non vengono scelti prodotti, si attende il permesso del supervisore per uscire; altrimenti, si viene serviti da un cassiere, con tempi proporzionali al numero di prodotti acquistati.

Al termine degli acquisti, il client chiude la socket e termina il programma.

1.3 SERVER

Il server rappresenta il cuore del supermercato e si occupa della gestione delle operazioni principali.

I client possono connettersi al server tramite socket, con l'indirizzo e la porta configurati nel file **"parameters/parameters.h"** attraverso le costanti **SERVER_ADDRESS** (predefinito localhost) e **PORT** (predefinito 9090).

Alcuni parametri del supermercato, definiti sempre in **"parameters.h"**, includono:

- MAX_CASHIERS: numero massimo di cassieri,
- ACTIVE_CASHIERS: numero di cassieri attivi,
- MAX_CLIENTS: numero massimo di clienti che possono essere presenti contemporaneamente,
- CLIENTS_BATCH_SIZE numero di clienti che possono entrare in un lotto quando il numero di clienti scende sotto la soglia MAX_CLIENT - CLIENTS_BATCH_SIZE.

Il server utilizza delle strutture dati a coda di tipo FIFO thread-safe, come:

- waiting_to_enter: coda dei clienti in attesa di entrare,
- clients: coda dei clienti attivi nel supermercato,
- waiting_to_exit: coda dei clienti che non hanno fatto acquisti, in attesa del permesso per uscire.

Quando il server viene avviato, inizializza i cassieri e il supervisore, creando per ciascuno un thread dedicato per la gestione della logica, con tempi di servizio casuali e le loro rispettive code. Successivamente, viene effettuato il bind della socket ed il server entra nel loop principale, mettendosi in ascolto di connessioni client con la funzione listen.

Ogni volta che un client si connette, viene creato un nuovo thread che lo gestisce tramite la funzione void* handleClient(void* arg) . Il thread verifica se il numero di clienti attivi non supera il limite massimo consentito; in caso contrario, il client viene messo in coda (waiting_to_enter) finché non è possibile farlo entrare. Una volta all'interno, il server invia la lista dei prodotti disponibili e attende il carrello del cliente. Se il carrello contiene prodotti, il cliente viene accodato al cassiere con meno clienti tramite get_shortest_queue; altrimenti, viene inserito nella coda del supervisore (waiting_to_exit).

I thread dei cassieri, che eseguono la funzione cashier_logic, operano in loop e gestiscono i clienti in coda calcolando il tempo necessario per completare il pagamento. Una volta servito un cliente, il cassiere lo rimuove dalla coda, lo notifica e chiude la connessione. Se sono soddisfatte le condizioni per l'ingresso di nuovi clienti, i cassieri risvegliano i thread in attesa.

Il supervisore segue una logica simile ai cassieri, gestendo i clienti senza acquisti con tempi di servizio casuali definiti dalla funzione rand.

1.4 DETTAGLI IMPLEMENTATIVI

Il server utilizza una gestione multithreading per garantire il funzionamento parallelo di cassieri, supervisori e clienti.

THREADS E MULTITHREADING

I thread sono stati preferiti ai processi per gestire i cassieri, i clienti e il supervisore per diversi motivi. Innanzitutto, i thread condividono lo stesso spazio di memoria, permettendo una comunicazione più semplice tra i componenti del server e riducendo la complessità di gestione rispetto ai processi.

Inoltre, la gestione multithread consente una sincronizzazione più intuitiva e flessibile, tramite primitive come mutex e condition variables, evitando la duplicazione di risorse per gestire meglio un elevato numero di clienti e operazioni in parallelo.

GESTIONE DEI THREAD E SINCRONIZZAZIONE

La gestione multithread del server è stata implementata utilizzando la libreria **pthread**, con particolare attenzione alla sincronizzazione dei thread per evitare problemi di concorrenza. Le primitive utilizzate includono:

- **pthread_mutex_t**: per bloccare l'accesso alle risorse condivise,
- **pthread_cond_t**: per sospendere e risvegliare thread in modo controllato, permettendo ad esempio di notificare ai thread in attesa che un cassiere si è liberato.

Per gestire l'ingresso di nuovi clienti, si utilizza **pthread_cond_wait** per sospendere i thread fino a quando le condizioni per l'ingresso non sono soddisfatte, e **pthread_cond_broadcast** per notificare quando i clienti possono entrare.

ALLOCAZIONE DELLE RISORSE E GESTIONE DELLA CONCORRENZA

La gestione delle risorse del server (client e cassieri) è progettata per essere efficiente e scalabile. Ogni volta che un nuovo cliente si connette, viene creato un thread separato per gestirlo. Il carico viene distribuito equamente tra i cassieri utilizzando un algoritmo per individuare la coda con il minor numero di clienti, riducendo i tempi di attesa complessivi.

Il server gestisce ogni client in modo indipendente, il che permette una gestione parallela delle operazioni e una maggiore efficienza. Questa scelta garantisce che il sistema possa gestire un numero elevato di clienti senza rallentamenti significativi.

LOGICA DI GESTIONE DEL FLUSSO DEI CLIENTI

Il server è responsabile della gestione dinamica dei clienti e della loro interazione con il supermercato. La logica che regola l'afflusso dei clienti è gestita tramite una serie di condizioni definite, che controllano quando e come i clienti possono entrare e uscire.

Per ottimizzare il flusso, abbiamo definito il parametro **CLIENTS_BATCH_SIZE**. Ogni volta che il numero di clienti attivi scende al di sotto di una certa soglia, il sistema permette l'ingresso di un nuovo gruppo di clienti. Questo migliora la gestione delle risorse e riduce la complessità di sincronizzazione tra i vari thread.

SUPERVISORE E GESTIONE DEI CLIENTI SENZA ACQUISTI

Un aspetto particolare del sistema è la gestione dei clienti che non effettuano acquisti. Quando un cliente esce dal supermercato senza aver comprato nulla, viene gestito dal supervisore, un thread separato con una logica simile ai cassieri. Il supervisore verifica la coda **waiting_to_exit** e, dopo un intervallo di tempo casuale, permette ai clienti di uscire. Questo aggiunge realismo alla simulazione, trattando i clienti che non hanno completato un acquisto in modo differente rispetto a quelli che devono essere serviti alle casse.

CAPITOLO 2 – BUILD ED ESECUZIONE DEL SOFTWARE

2.1 GUIDA ALLA COMPILAZIONE DEL PROGETTO

Per compilare il progetto è necessario avere i seguenti strumenti:

- GCC (GNU Compiler Collection),
- Make,
- Bash,
- Docker,
- Docker Compose.

2.2 ISTRUZIONI PER LA COMPILAZIONE

PER ESEGUIRE IL PROGRAMMA SULLA PROPRIA MACCHINA:

CLONARE LA REPOSITORY

Per iniziare, bisogna clonare la repository sulla propria macchina. Aprire il terminale ed eseguire:

```
git clone https://github.com/leokm02/LSO-23-24.git
cd LSO-23-24
```

COMPILARE IL PROGRAMMA

Il progetto include un Makefile che gestisce l'intero processo di compilazione. Per compilare il programma eseguire il seguente comando:

```
make
```

PER ESEGUIRE IL PROGETTO IN UN CONTAINER DOCKER:

CLONA LA REPOSITORY

Innanzitutto, clona la repository sulla tua macchina. Aprire il terminale ed eseguire:

```
git clone https://github.com/leokm02/LSO-23-24.git
cd LSO-23-24
```

Se possiedi già una copia del progetto è possibile saltare questo passaggio.

ESECUZIONE PROGRAMMA

All'interno del progetto, lo script "./launchApp.sh" gestisce la fase di build e di esecuzione. Per lanciare il programma, eseguire:

```
./launchApp.sh
```

INTERROMPERE IL PROGRAMMA

Per interrompere l'esecuzione del server premere CTRL + C.

Questo arresterà il server e salverà un file log.txt contenente il log del server nella cartella del progetto.

2.3 ISTRUZIONI PER L'ESECUZIONE

Per avviare il server, eseguire i seguenti comandi dal terminale:

```
./server.out
```

Se il server è già in esecuzione in un container Docker è possibile saltare questo passaggio.

PER AVVIARE UN CLIENT BOT IN MODALITÀ AUTOMATICA:

(Questo avvierà un client bot che sceglierà i prodotti in un tempo casuale e li comprerà.)

./client.out

PER AVVIARE IL CLIENT IN MODALITÀ INTERATTIVA:

./client.out -i

Per generare più client automatici contemporaneamente, nel progetto è incluso lo script *launchMultipleClient.sh* che genera multipli clienti bot allo stesso tempo. Per utilizzarlo eseguire il seguente comando:

./launchMultipleClient.sh CLIENTS_TO_LAUNCH

2.4 RISOLUZIONE DEI PROBLEMI

• **Permission Denied**: Se viene visualizzato un errore di permessi durante l'esecuzione degli script, è possibile risolverlo rendendo gli script eseguibili:

chmod +x launchApp.sh launchMultipleClient.sh

• **Missing Dependencies**: Assicurarsi che tutte le dipendenze siano correttamente installate. Utilizzare il package manager appropriato (ad esempio apt, yum, brew) per installare eventuali pacchetti mancanti.