PROJET NF06 - Système FeedBoof

Gestion des commandes de restaurant en langage C

Elouan Deschamps & Pierre Lenicolais

Semestre A25

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

I - Sommaire

II - Glossaire

- **FIFO (First In, First Out)** : Principe de gestion de file d'attente où le premier élément inséré est le premier traité.
- **CSV (Comma-Separated Values)** : Format de fichier utilisé pour le stockage et la lecture des données restaurants.
- **Liste Chaînée** : Structure de données dynamique composée de nœuds reliés par des pointeurs, utilisée ici pour la file d'attente.
- **CLI (Command Line Interface)** : Interface textuelle permettant l'interaction avec l'utilisateur via la console.
- **Pointeur** : Variable stockant l'adresse mémoire d'une autre variable, essentielle pour naviguer dans la liste chaînée.
- **Allocation Dynamique** : Gestion de la mémoire en temps d'exécution (via `malloc`), utilisée pour créer de nouveaux nœuds de commande.
- **Git / GitHub** : Git est un système de contrôle de version permettant de suivre les modifications du code source. GitHub est la plateforme collaborative utilisée pour synchroniser le travail entre les membres du binôme.

III - Introduction

Dans le cadre de l'unité d'enseignement NF06 (Pratique de la programmation) à l'Université de Technologie de Troyes, nous avons conçu et réalisé FeedBoof, un système complet de gestion de commandes pour restaurants en langage C.

Le projet a deux grandes parties : l'importation de données structurées depuis des fichiers CSV et la gestion des commandes via une file d'attente FIFO implémentée par une liste chaînée.

Ce travail a été réalisé en binôme par Elouan Deschamps et Pierre Lenicolais. La répartition des tâches est la suivante : Pierre s'est concentré sur la définition des structures de données, le moteur de lecture CSV ainsi que de l'interface utilisateur (CLI) pour les profils Client et Admin. Elouan a pris en charge l'implémentation de la logique de la file d'attente, le scindage du code en plusieurs modules et la résolution des bugs restants.

IV - Architecture et Structures de données

Cette section détaille les structures de données utilisés pour le système FeedBoof.

Structures des Restaurants et Menus

Pour stocker les informations permanentes (lues depuis le fichier CSV), nous avons utilisé des structures imbriquées permettant de regrouper les données par restaurant.

- **Structure** `MenuItem` : Elle représente un plat individuel. Chaque item possède une catégorie (Entrée, Plat, Dessert, Boisson), un nom, un prix en float et un entier pour le stock (gérant la valeur -1 pour le stock illimité).

- **Structure** `Restaurant` : Elle regroupe un nom, un compteur du nombre de plats effectifs (`count`) et un tableau fixe de 15 structures `MenuItem`.

Structure de la file d'attente des commandes

Contrairement aux menus, le nombre de commandes en attente est imprévisible. Nous devons donc utiliser une structure de liste chaînée pour implémenter la file d'attente.

- **Structure** `Commande` : Chaque nœud de la liste contient le nom du restaurant concerné, un numéro de commande unique et croissant, ainsi que le montant total. Un pointeur `*next` permet de lier dynamiquement les commandes entre elles.

V - Gestion du fichier CSV

Le stockage des données des restaurants et de leurs menus est assuré par un fichier externe au format CSV.

Structure du fichier CSV

Le fichier `restaurants.csv` suit une structure hybride pour optimiser la lecture. Les noms des restaurants servent de délimiteurs de section, suivis des plats . Chaque ligne de plat respecte ce format : `Catégorie, Nom du plat, Prix, Stock`

Nous avons rajouté au fichier initial deux établissements supplémentaires : un restaurant Indien et un restaurant Français.

Algorithme de lecture du CSV

Le code de lecture repose sur une analyse ligne par ligne via la fonction `fgets`. L'algorithme distingue deux types de lignes :

- **Lignes de restaurant** : Identifiées par l'absence de virgule. Le texte est alors copié dans le champ `nom` d'une nouvelle structure `Restaurant`.
- **Lignes de menu** : Découpées en segments (tokens) grâce à `strtok`.

Pour garantir d'avoir les bons formats, nous utilisons `atof` pour convertir les prix en nombres flottants et `atoi` pour transformer les stocks en entiers.

Choix techniques

Lors de la création du lecteur, nous avons dû faire face à plusieurs défis techniques pour assurer le bon fonctionnement du programme.

- **Gestion des espaces** : Un nettoyage est effectué après chaque virgule pour éviter que des espaces ne soient inclus dans les noms des plats.
- **Sécurité des formats** : L'utilisation de `strcspn` permet de supprimer proprement le caractère de saut de ligne (`\n`) en fin de chaîne
- **CSV inchangé** : Toute commande validée décrémente le stock, mais le fichier CSV ne bouge pas.

VI - Logique FIFO et Traitement des Commandes

FeedBoof repose sur la gestion des commandes via une file d'attente de type FIFO (First In, First Out).

Principes de la file d'attente

Pour implémenter cette file, nous avons utilisé une liste chaînée dynamique. Contrairement à un tableau, cette structure nous permet de libérer la mémoire dès qu'une commande est servie (opération `dequeue`).

Trois fonctions implémentées par Elouan dans le module `fifo.c`, régissent ce système :

- `isEmpty()` : Vérifie si la file contient des commandes.
- `enqueue()` : Ajoute une commande à la fin de la file (en queue).
- `dequeue()` : Retire la commande la plus ancienne (en tête) pour la traiter.

Fonctionnement d'une commande

Le traitement d'une commande suit un parcours prédéterminé entre les deux profils du système :

- **Génération** : Lorsque le client valide son panier, une structure `Commande` est créée dynamiquement avec un numéro unique.
- **Enfilage (`enqueue`)** : La commande est placée à la suite des autres. Si la file est vide, elle devient la tête de liste.
- **Attente** : Elle reste en mémoire tant que l'administrateur n'a pas validé le service des commandes précédentes.
- **Défilage (`dequeue`)** : L'administrateur retire la tête de file. À ce moment, les statistiques de ventes du restaurant concerné sont mises à jour et la mémoire du nœud est libérée via la fonction `free()`.

VII - Interface Utilisateur et Profils

L'interface utilisateur a été conçue comme une Interface en Ligne de Commande (CLI) séparé selon le profil client et Administrateur.

Architecture des Menus

Le programme est fait autour d'un menu principal permettant de basculer entre les deux profils. Chaque profil fonctionne de manière autonome :

- **Le Profil Client** : Dédié à la prise de commande.
- **Le Profil Administrateur** : Dédié à la gestion de commande et les statistiques.

Le Profil Client

Le parcours client a été fait de manière simple pour limiter les erreurs de navigation. Le processus se décompose en trois étapes :

- **Sélection de l'établissement** : Affichage dynamique de la liste des restaurants chargés depuis le CSV.
- **Consultation du menu** : Présentation détaillée des plats avec affichage en temps réel du stock restant.
- **Validation** : Vérification de la disponibilité avant l'envoi de la commande dans la file d'attente globale.

Le Profil Administrateur

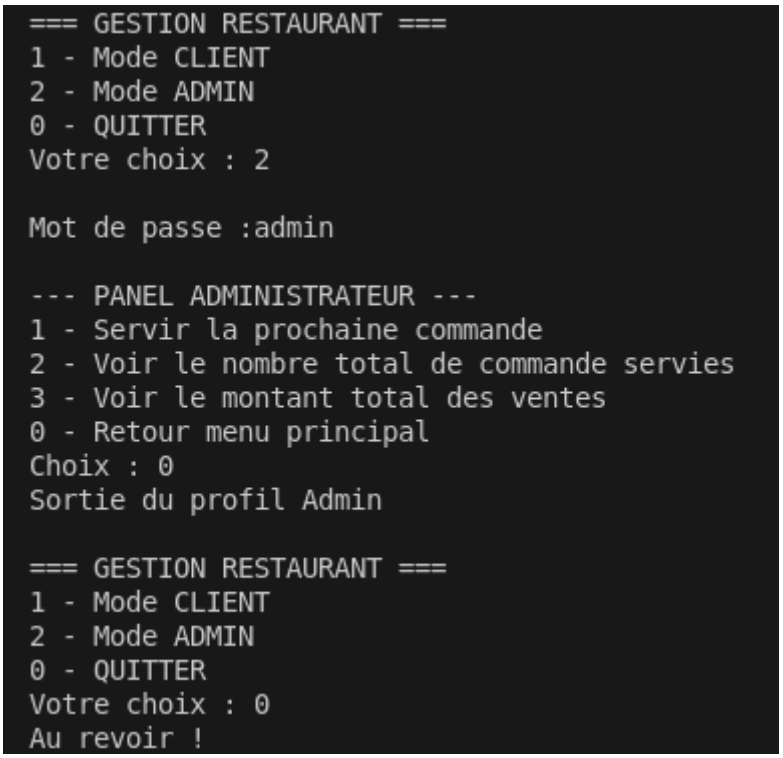
Ce profil offre une vue d'ensemble sur l'activité du système. Ses fonctions sont :

- **Traitement des commandes** : L'administrateur visualise la tête de la file d'attente et valide le service, ce qui déclenche le défilage (`dequeue`) et la libération de la mémoire.
- **Suivi des ventes** : Un module de statistiques permet de consulter le chiffre d'affaires cumulé par restaurant, offrant ainsi une visibilité sur la rentabilité de chaque établissement.

Prévention des erreurs de saisie

Pour prévenir tout plantage du programme dû à une mauvaise manipulation, nous avons pris des précautions :

- **Filtrage des choix** : Chaque menu est protégé par une structure `switch` incluant un cas `default`. Si l'utilisateur saisit un nombre hors des options proposées, un message d'erreur est affiché et le menu est rechargé.
- **Validation des stocks** : Le système empêche techniquement la commande d'un plat dont le stock est épuisé (0), informant l'utilisateur de l'indisponibilité.



VIII - Organisation des fichiers et modularité

Pour avoir un projet clair, nous avons fait le choix de ne pas travailler sur un fichier unique, mais d'utiliser plusieurs fichiers en séparant les déclarations (`.h`) des implémentations (`.c`).

Organisation des fichiers

Le projet est structuré en trois modules distincts, chacun ayant une responsabilité précise :

- **Module FIFO (`fifo.h` / `fifo.c`)** : Ce module est une bibliothèque générique gérant la structure de la liste chaînée. Il contient les fonctions d'enfilage et de défilage.
- **Module Fonctions (`fonctions.h` / `fonctions.c`)** : Il regroupe le chargement du CSV, la gestion des stocks et les statistiques.
- **Point d'entrée (`main.c`)** : Il appel les autres modules gère la boucle principale de l'interface utilisateur.

Justification de la modularité

Bien que le scindage du code ait ajouté une complexité initiale, ce choix a plusieurs avantage :

- **Travail collaboratif** : L'un pouvait travailler sur l'optimisation de la file d'attente pendant que l'autre finalisait l'interface, sans risque de conflits majeurs dans le code.
- **Lisibilité** : Le `main.c` reste court et compréhensible, chaque fonction étant rangée par thématique.
- **Professionnalisme** : Cette structure se rapproche des standards de développement industriel, un aspect que nous souhaitions intégrer à notre rendu.

IX - Conclusion

Le projet FeedBoof répond à l'ensemble du cahier des charges du projet NF06. Nous avons réussi à mettre en place un système complet où la gestion des stocks et la file d'attente FIFO fonctionnent de manière synchronisée.

Ce travail nous a permis de passer d'exercices isolés, vus en cours ou en TP, à un projet global mettant en application l'ensemble des connaissances acquises. La séparation du code en plusieurs modules a été un choix payant, facilitant notre collaboration en binôme et garantissant la clarté du programme final. Ce projet nous a également permis l'apprentissage de l'outil de versionnage Git (via la plateforme GitHub), un outil de gestion de version essentiel que nous avons appris à manipuler pour synchroniser nos travaux.

Le développement de ce logiciel a été très formateur et enrichissant. Partir de zéro pour arriver à ce résultat fonctionnel est gratifiant et nous donne une vision concrète de ce que permet la programmation en langage C.