RAPPORT DU TP4-CSR

Auteurs :

- **AKA AKA JEAN RUPHIN EMMANUEL**
- **FASSASSI AMIRATH**

Sommaire

I. Introduction

Ce projet vise à simuler le fonctionnement d'un restaurant pour améliorer la fluidité de son service en utilisant des techniques de programmation multithreadée. L'objectif est de modéliser les interactions entre les clients et les employés, ainsi que de gérer les conflits d'accès aux ressources partagées. La simulation inclut les éléments suivants : un restaurant avec une capacité limitée de 25 places, un buffet avec plusieurs compartiments, des clients, un employé pour le réapprovisionnement, un stand de cuisson et un cuisinier.

II. Description des Composants

1. Restaurant

- **Nombre de places limité à 25** : Le restaurant peut accueillir un maximum de 25 clients à la fois. Les clients supplémentaires attendent à l'extérieur si le restaurant est plein.
- **Initialisation** : Le restaurant initialise les compartiments du buffet, les clients, l'employé et le stand de cuisson.

2. Compartiments du Buffet

- **Quatre compartiments** : Poisson cru, viande crue, légumes crus, nouilles froides. Chaque compartiment contient initialement 1 kg de nourriture.
- **Accès concurrent** : Les clients et l'employé peuvent accéder au buffet simultanément, mais pas au même compartiment en même temps. La synchronisation est gérée pour éviter les conflits d'accès.

3. Employé du Buffet

- **Réapprovisionnement** : L'employé vérifie régulièrement les niveaux de nourriture et réapprovisionne les compartiments lorsque le stock tombe en dessous de 100 g. L'employé n'est jamais à court de nourriture.

4. Stand de Cuisson et Cuisinier

- **Cuisson** : Les clients doivent passer au stand de cuisson après avoir rempli leur assiette. Le cuisinier cuit les plats un par un, sans nécessairement respecter l'ordre d'arrivée des clients.

5. Clients

- **Cycle des clients** : Entrer dans le restaurant, se servir au buffet, attendre la cuisson, manger, puis sortir. Chaque client suit ce cycle de manière autonome en tant que thread.

III. Choix d'Implémentation

1. Utilisation de Threads

- **Classe Client** : Chaque client est un thread indépendant, permettant de simuler des actions concurrentes telles que se servir au buffet et attendre la cuisson.
- **Classes Employé et Cuisinier** : Ces classes sont également implémentées en tant que threads pour gérer les opérations de réapprovisionnement et de cuisson.
- **La classe compartiment :** a été choisi comme moniteur car les operations devaientr se synchronisées au niveau de chaque compartiment. Si on utilisait une classe buffet comme moniteur en lieu et place de Compartiment,cela serait trop contraignant à gérer
- **La classe Stand de cuisson :** a aussi été choisie comme moniteur pour pouvoir synchroniser les opérations de cuisson. Nous avons aussi ajouté une autre opération demander cuisson pour pouvoir mieux gérer la manière dont le client reçoit les plats des clients et comment il les traite. Les clients quittent le buffet pour se rendre au stand de cuisson et font une demande pour pouvoir cuire leurs plats. Et c’est à l’ordonnanceur d’attribuer au cuisinier, de façon aléatoire peut importe l’ordre d’arriver des clients, le plat qui serai traité.

2. Synchronisation

- **Méthode synchronized** : Utilisée pour gérer l'accès concurrent aux compartiments et au stand de cuisson, évitant les conflits et garantissant la sécurité des threads.
- **wait() et notify() / notifyAll()** : Utilisés pour gérer les attentes et les notifications entre les clients, l'employé et le cuisinier.

IV. Problèmes de Synchronisation et Solutions

1. Accès aux Compartiments

- **Problème** : Concurrence entre clients pour accéder au même compartiment.
- **Solution** : Utilisation de `synchronized` sur les méthodes `prendreNourriture` et `approvisionner` pour assurer l'accès exclusif. `wait()` et `notifyAll()` sont utilisés pour gérer les attentes des clients lorsque le stock est insuffisant.

2. Gestion du Stock

- **Problème** : Réapprovisionnement pendant que les clients se servent.
- **Solution** : L'employé attend que le compartiment soit libre avant de le réapprovisionner, utilisant `wait()` et `notifyAll()`. Cela évite les conflits entre le réapprovisionnement et le service des clients.

3. Stand de Cuisson

- **Problème** : Synchronisation entre les clients et le cuisinier.
- **Solution** : Utilisation de `synchronized` sur les méthodes du stand de cuisson. Les clients utilisent `wait()` lorsqu'ils demandent la cuisson, et le cuisinier utilise `notifyAll()` après avoir terminé la cuisson pour s'assurer que les clients et le cuisinier sont correctement synchronisés.

4. Gestion des Places dans le Restaurant

- **Problème** : Concurrence pour entrer dans le restaurant.
- **Solution** : Verrou sur la classe `Restaurant` pour gérer l'attente et l'entrée des clients. `wait()` et `notify()` sont utilisés pour gérer les files d'attente et l'occupation des places.

V. Conclusion

Ce projet simule efficacement le fonctionnement concurrentiel d'un restaurant en utilisant des techniques de programmation multithreadée et des mécanismes de synchronisation. Les choix d'implémentation assurent une fluidité et une sécurité des opérations, permettant une gestion efficace des ressources partagées et une meilleure expérience pour les clients.