Programmation concurrente : Laboratoire automate

Découpage du problème et ressources à protéger

Il y a trois tâches principales qui tournent en concurrence avec le thread main :

- MONNAIE : s'occupe de l'insertion des pièces ;
- DISTRIBUTEUR : s'occupe de l'exécution de la commande ;
- VENDEUR : gère le solde, les bouteilles et les actions utilisateur.

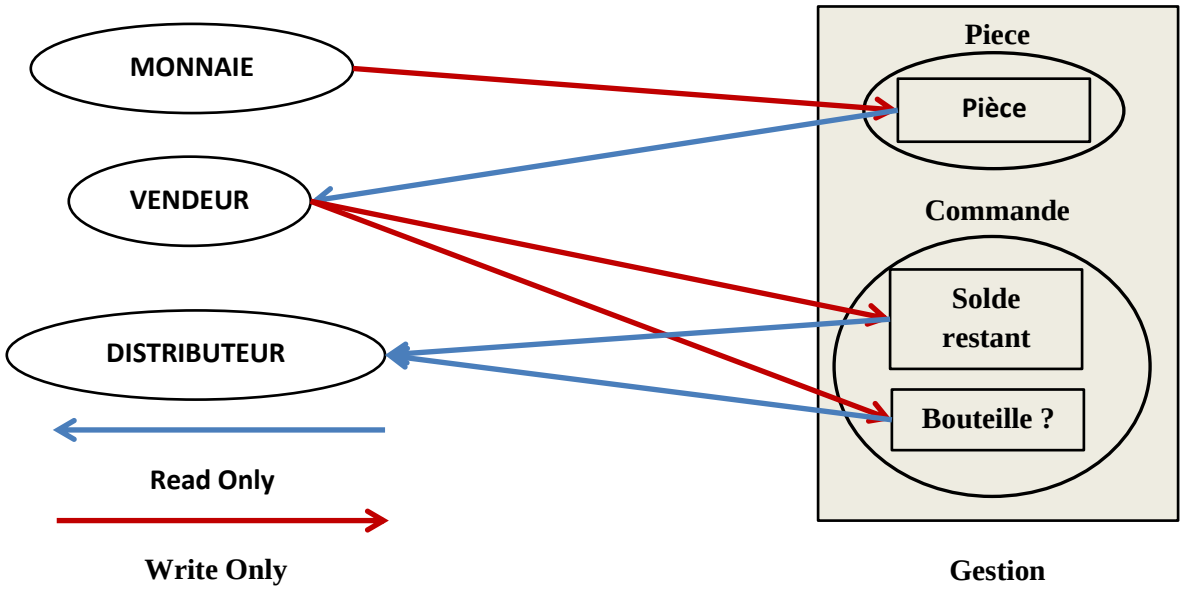
La tâche VENDEUR contient les actions permettant de quitter le programme, lancer l'insertion de pièces ou exécuter la commande. Lorsque l'insertion de la monnaie est terminée, VENDEUR va mettre à jour le solde selon la pièce insérée.

Lorsque l'exécution de la commande est lancée, VENDEUR indique à DISTRIBUTION si l'utilisateur a assez de crédit pour se payer une bouteille et si le stock de bouteilles est suffisant.

La tâche MONNAIE indique à VENDEUR la pièce insérée.

La tâche DISTRIBUTEUR donne à l'utilisateur une bouteille si cela est possible, sinon il lui annonce l'impossibilité de satisfaire sa demande.

En termes de ressources à protéger, on peut résumer avec le schéma suivant :



Implémentation

Nous avons découpé le problème en 5 fichiers :

- main.cpp : Permet de lancer l'automate
- structures.h/structures.cpp : Contient la définition et l'implémentation des classes qui seront utilisées pour la communication entre les différentes tâches.
- automate.h/automate.cpp : Contient la définition et l'implémentation des différentes fonctions utiles à l'interaction avec l'automate.

Dans le fichier *main.cpp*, nous lançons l'automate avec la fonction *startAutomate()*.

Les fichiers *structures.h/structures.cpp* contiennent les définitions et les implémentations des classes qui seront utilisées pour la communication entre les différents threads. Les classes sont les suivantes :

- **Piece** : contient un attribut privé *valeur* qui représente la pièce insérée par l'utilisateur. Elle possède des méthodes permettant d'obtenir cette valeur ou de la modifier.
- **Commande** : contient deux attributs privés : *solde* et *bouteille.* Le premier contient la somme des pièces insérées dans la machine, non encore dépensées. Le deuxième indique au système si les conditions permettant l'octroi d'une bouteille à l'utilisateur sont réunies.
  Si le solde est suffisant et qu'il reste assez de bouteilles, une bouteille sera prélevée du stock et le solde de l'utilisateur sera modifié. Dans le cas contraire, le système indique à l'utilisateur qu'il ne peut pas lui vendre une bouteille.
  La classe possède deux méthodes : une permettant de récupérer la valeur du solde et une autre qui permet de savoir si les conditions d'obtention d'une bouteille sont réunies. Deux autres méthodes permettent de mettre à jour le solde et mettre à jour le drapeau *bouteille*.
- **Gestion** : contient deux attributs privés, de type *Piece* et *Commande*, permettant ainsi au thread VENDEUR de communiquer avec les autres threads[1].

Dans chacune des méthodes permettant de modifier une valeur, il est vérifié si le thread a le droit d'effectuer l'action. Cette vérification est faite à l'aide d'identificateurs de thread définis à l'aide de la macro *#define*.

Dans les fichiers *automate.h/automate.cpp* se trouvent les fonctions relative à la gestion de l'automate ainsi que les constantes globales utilisées pour associer une touche du clavier à une action de l'automate. D'autres constantes sont définies, notamment pour fixer le nombre de bouteilles en stock à l'allumage de l'automate et leur prix. Finalement, le code de retour du thread VENDEUR est spécifié.

D'autres variables non-constantes sont utilisées, il s'agit du *mutex* qui sera utilisé entre les différents threads ainsi que 3 variables conditions permettant de réveiller les threads au moment opportun.

[1] Pour plus d'informations sur la communication entre les threads, se référer au chapitre « Découpage du problème et ressources à protéger ».

Les fonctions principales sont :

- **MONNAIE** : fonction s'occupant de l'insertion des pièces. Attend principalement sur la variable de condition *inserer*.
- **DISTRIBUTEUR** : fonction s'occupant de l'affichage de la commande. Attend principalement sur la variable de condition *distribuer*.
- **VENDEUR** : fonction principale, allant appeler les fonctions *MONNAIE* et *DISTRIBUTEUR* en leur envoyant le signal correspondant, puis va attendre sur la variable de condition *vendeur* que l'action soit terminée.

Pour ces 3 fonctions principales, puisque le programme doit tourner en concurrence, il faut effectuer dans chacune d'entre-elle une boucle infinie. Dans cette boucle, le thread attendra son tour selon la variable de condition correspondante.

Un point essentiel pour les fonctions *MONNAIE* et *DISTRIBUTEUR* : il faut définir la possibilité de terminaison d'un thread depuis l'extérieur ainsi que son type de terminaison. Le type de terminaison est de type asynchrone.

La fonction *startAutomate()*, appelé depuis le thread *main* permet les mutex et les variables de conditions par un appel à la méthode *initializeMutexCond()*. Ensuite vient la création des variables qui seront utilisées pour la communication entre les threads. Nous créons ensuite les threads.

Le thread main attendra uniquement sur le thread VENDEUR car c'est lui qui va définir si les autres threads peuvent arrêter de s'exécuter. Une fois le thread VENDEUR terminé, le programme vérifie que le code de retour correspond à celui précisé dans l'en-tête et si c'est le cas, il va lancer la terminaison des threads MONNAIE et DISTRIBUTEUR. La dernière étape effectuée est l'appel de la méthode *destroyMutexCond()*, qui s'occupera de détruire les mutex et les variables de conditions.

Vérification fonctionnement

Pendant que l'utilisateur effectue son choix d'action, nous connaissons le solde actuel, le nombre de bouteilles restantes ainsi que leur prix unitaire. A chaque insertion de monnaie, nous pouvons essayer d'obtenir une bouteille. Dans le cas où le solde n'est pas suffisant ou qu'il n'en reste plus en stock, le client ne recevra pas de bouteille. Dans le cas contraire, le client recevra une bouteille, le montant de celle-ci sera déduit du solde et le stock diminuera.

Concernant la comparaison des pièces acceptées, puisqu'il s'agit de valeurs réelles, une comparaison utilisant l'opérateur == n'est pas suffisante. Dans ce cas, la comparaison entre 2 réels doit se faire avec une tolérance notée epsilon. Dans notre cas, $\varepsilon = 10^{-3}$, car les décimales au-delà de la deuxième ne sont pas significatives pour des centimes.