Problème des philosophes

Assala ASSELLALOU

21 mars 2025

Résumé

Ce rapport du TP1 explore des solutions pour permettre à des philosophes de partager équitablement des ressources, ici représentées par des fourchettes, sans se bloquer mutuellement. Des approches variées sont testées pour éviter les situations où certains philosophes ne peuvent plus manger. L'objectif est d'assurer un accès ordonné et équilibré aux fourchettes pour tous.

1 Introduction

N philosophes sont autour d'une table. Il y a une assiette par philosophe, et *une* fourchette entre chaque assiette. Pour manger, un philosophe doit utiliser les deux fourchettes adjacentes à son assiette (et celles-là seulement).

Un philosophe peut être dans l'état :

— penseur : il n'utilise pas de fourchettes ;

— mangeur : il utilise les deux fourchettes adjacentes ; aucun de ses voisins ne peut manger ;

— demandeur : il souhaite manger mais ne dispose pas des deux fourchettes.

Visuellement les états mangeur/demandeur/penseur sont représentés par un rond noir (ou rouge en cas de possible problème)/un rond blanc/rien.

2 Première approche : les fourchettes sont des ressources critiques

2.1 Stratègie de base : Philo1.java

Cette solution consiste à définir *fourchettes* un tableau de sémaphores initialisées a 1 représentant les fourchettes des philosophes dont le nombre est égale à *nbPhilosophes,* le nombre de philosophes sur table. La méthode *demanderFourchettes* consiste à prendre la fourchette droite, ensuite la fourchette gauche. Après, la méthode *libererFourchettes* consiste à libérer les fourchettes : gauche puis droite.

2.2 Problème d'interblocage

Néanmoins, cette solution peut rapidement faire preuve d'interblocage. Cette situation bloquante se produit lorsque tous les philosophes demandent les fourchettes et donc au moment où ils prennent tous la fourchette de droite.

Un moyen simple pour cela est d'introduire une temporisation suffisante entre les prises de fourchette : Thread.sleep(500) .

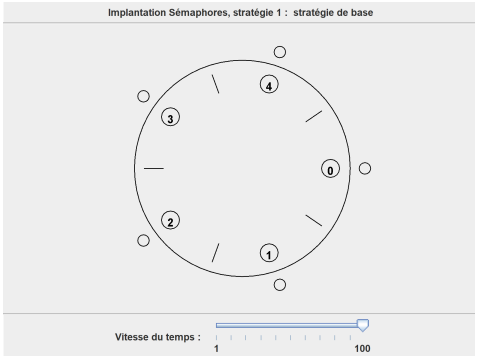

FIGURE 1 – Mise en évidence de la situation d'interblocage

2.3 Adaptations de la version de base

2.3.1 Adaptation : Philo2.java

Dans cette adaptation, lorsqu'un philosophe, notamment le philosophe 0, souhaite manger, il prend d'abord la fourchette de droite puis la fourchette de gauche. Pour les autres philosophes, l'ordre est inversé : ils prennent d'abord la fourchette de gauche ensuite celle de droite. Cette inversion d'ordre permet de réduire la probabilité qu'un ensemble de philosophes se retrouve dans une situation d'attente circulaire.

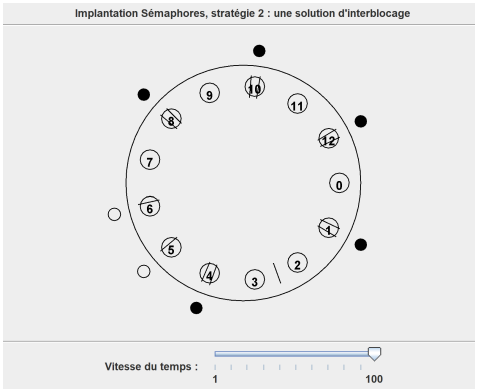

FIGURE 2 – adaptation de la situation d'interblocage

2.3.2 Adaptation : Philo3.java

Une autre adaptation est lorsqu'un philosophe souhaite manger, il tente d'acquérir d'abord la fourchette de droite. Ensuite, il essaie d'acquérir la fourchette de gauche à l'aide de la méthode `tryAcquire()`, qui permet de tenter d'obtenir la ressource. Si la fourchette de gauche n'est pas disponible, le philosophe relâche la fourchette de droite et réessaie d'acquérir les fourchettes dans le prochain cycle.

```java
public void demanderFourchettes (int no) throws InterruptedException{
    boolean mange = false;
    while (!mange) {

        int fd = Main.FourchetteDroite(no);
        int fg = Main.FourchetteGauche(no);

        fourchettes[fd].acquire();

        if (fourchettes[fg].tryAcquire()) {

            IHMPhilo.poser(fd, EtatFourchette.AssietteGauche);//Pour eviter qu'une seule fourchette soit sur l'assiette
            IHMPhilo.poser(fg, EtatFourchette.AssietteDroite);
            mange = true;

        } else {

            fourchettes[fd].release();

        }
    }
}
```

FIGURE 3 – Autre adaptation de la situation d'interblocage

3 Seconde approche : contrôler la progression d'un philosophe en fonction de l'état de ses voisins

3.1 Evaluation du degré de parallélisme : Philo4.java

Contrairement à la solution *Philo1.java* où les philosophes peuvent se bloquer mutuellement en attendant des fourchettes, la solution *Philo4.java* est optimale en termes de degré de parallélisme, car elle permet à un philosophe de manger immédiatement si ses voisins ne sont pas en train de manger.

Cette solution utilise des sémaphores et un suivi des états des philosophes pour garantir que seuls ceux qui peuvent manger le fassent. Cette gestion permet d'atteindre un degré de parallélisme pouvant atteindre la partie entière de **nbPhilosophes/2** dans le meilleur des cas.

3.2 Equité

3.2.1 Scénario démontrant la famine d'un philosophe

Dans ce scénario, P0 et P1 mangent pendant que P2 et P3 attendent. Lorsque P0 libère ses fourchettes, P3 tente de manger mais est bloqué par P1. Ensuite, lorsque P1 libère ses fourchettes, P2 peut demander à manger, mais si P0 et P1 recommencent à manger immédiatement, P2 et P3 restent bloqués dans une attente indéfinie, ce qui conduit à leur famine.

3.2.2 Solution avec gestion des priorités

Pour éviter la famine, on peut implémenter un système de priorités, par exemple, en assignant un ordre cyclique ou en favorisant le philosophe qui attend le plus longtemps. Cette solution permet à tous les philosophes de manger à tour de rôle et donc assure qu'aucun d'eux n'est bloqué indéfiniment.

Analyse de la solution

1. Degré de parallélisme dans le pire des cas : Dans cette approche, le degré de parallélisme est réduit par rapport à la solution optimale, car les philosophes doivent attendre leur tour en fonction de la priorité.

2. Attente maximum pour un philosophe demandeur : L'attente est courte, car chaque philosophe a un tour défini et ne peut être continuellement ignoré.

3. Limites de la solution proposée : Cette solution empêche la famine et assure que chaque philosophe aura un accès équitable aux ressources. Cependant, en imposant un ordre d'accès fixe cette solution limite le parallélisme ce qui ralentit l'accès des philosophes qui pouvaient manger simultanément dans la solution "optimale" .