TP3 : Problème des philosophes

Hamza Mouddene

25 octobre 2020

1 Première approche : les fourchettes sont des ressources critiques

— La version de base consiste à définir un tableau de sémaphores représentant n fourchettes où n est le nombre de philosophe, chaque fourchette de type sémaphore est initialisé à 1. Lors de chaque demande je prends la fourchette droite en premier puis la fourchette gauche, quand le philosophe finit de manger je repose les fourchettes dans le même ordre.

PhiloBase.java

```java
// Time-stamp: <08 déc 2009 08:30 queinnec@enseeiht.fr>

import java.util.concurrent.Semaphore;

public class PhiloBase implements StrategiePhilo {

    /** Attributs de classe */
    private static Semaphore[] fourchettes;

    /** Constructeur de classe */
    public PhiloBase (int nbPhilosophes) {
        this.fourchettes = new Semaphore[nbPhilosophes];
        for (int i = 0; i < nbPhilosophes; i++)
            this.fourchettes[i] = new Semaphore(1);
    }

    /** Le philosophe no demande les fourchettes.
     *  Précondition : il n'en possède aucune.
     *  Postcondition : quand cette méthode retourne, il possède les deux fourchettes adjacentes à son assiette. */
    public void demanderFourchettes (int no) throws InterruptedException{
        this.fourchettes[Main.FourchetteDroite(no)].acquire();
        IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.AssietteGauche);
        this.fourchettes[Main.FourchetteGauche(no)].acquire();
        IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.AssietteDroite);
    }

    /** Le philosophe no rend les fourchettes.
     *  Précondition : il possède les deux fourchettes adjacentes à son assiette.
     *  Postcondition : il n'en possède aucune. Les fourchettes peuvent être libres ou réattribuées à un autre philosophe. */
    public void libererFourchettes (int no) {
        this.fourchettes[Main.FourchetteDroite(no)].release();
        IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.Table);
        this.fourchettes[Main.FourchetteGauche(no)].release();
        IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.Table);
    }

    /** Nom de cette stratégie (pour la fenêtre d'affichage). */
    public String nom() {
        return "Implantation Sémaphores, stratégie de base";
    }
}
```

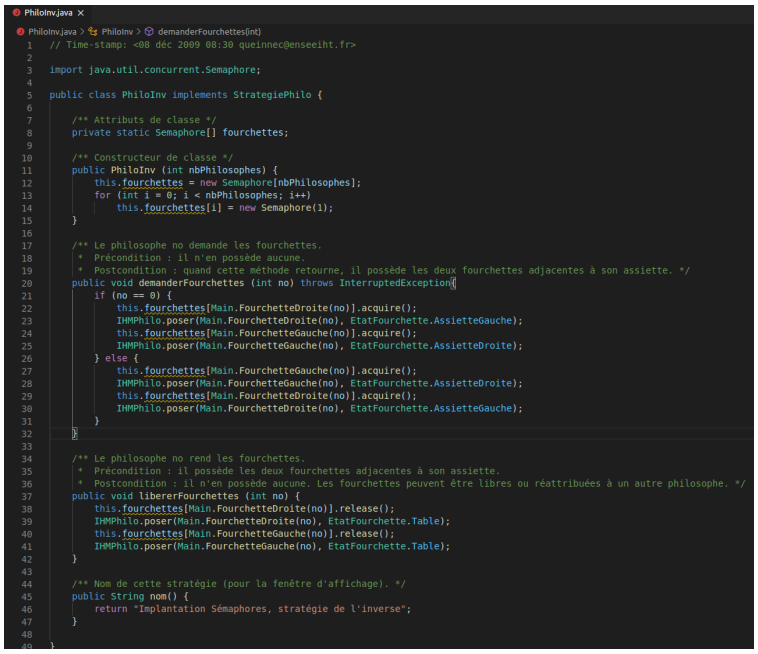
FIGURE 1 – Listing1 : PhiloBase.java

— Pour mettre en évidence la situation d'interblocage sur la version de base, il suffit de rajouter entre la prise de la fourchette droite est gauche Thread.sleep(1000). L'interblocage se produit au moment où tout les philosophes prennent la fourchette droite.

```java
/** Le philosophe no demande les fourchettes.
 *  Précondition : il n'en possède aucune.
 *  Postcondition : quand cette méthode retourne, il possède les deux fourchettes adjacentes à son assiette. */
public void demanderFourchettes (int no) throws InterruptedException{
    this.fourchettes[Main.FourchetteDroite(no)].acquire();
    IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.AssietteGauche);
    Thread.sleep(1000);
    this.fourchettes[Main.FourchetteGauche(no)].acquire();
    IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.AssietteDroite);
}
```

FIGURE 2 – Listing2 : Interblocage

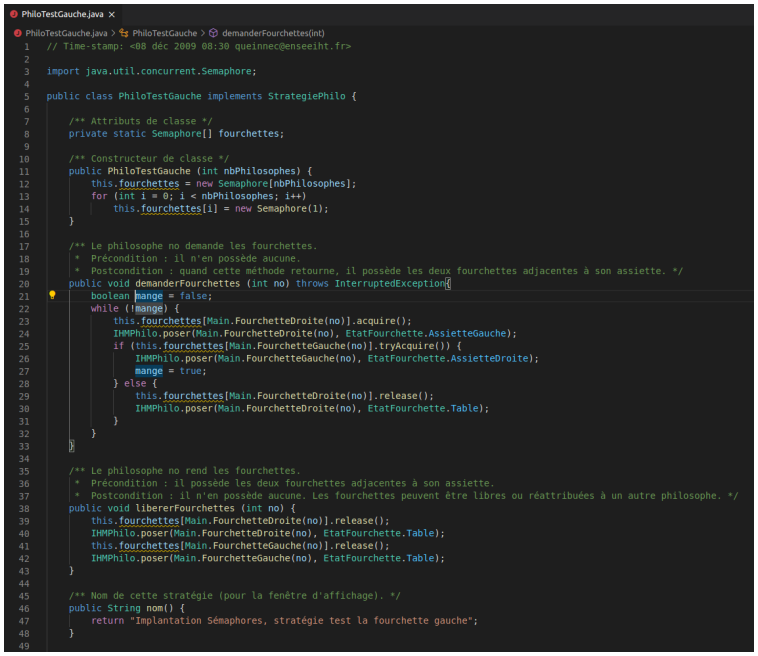
— Une première solution **(Vu en TD)** consiste à donner en premier la fourchette droite puis la fourchette gauche pour le premier philosophe, sinon elle inverse l'ordre de prise de fourchettes, cela évite la situation d'interblocage vue dans la question précedente.



```java
// Time-stamp: <08 déc 2009 08:30 queinnec@enseeiht.fr>

import java.util.concurrent.Semaphore;

public class PhiloInv implements StrategiePhilo {

    /** Attributs de classe */
    private static Semaphore[] fourchettes;

    /** Constructeur de classe */
    public PhiloInv (int nbPhilosophes) {
        this.fourchettes = new Semaphore[nbPhilosophes];
        for (int i = 0; i < nbPhilosophes; i++)
            this.fourchettes[i] = new Semaphore(1);
    }

    /** Le philosophe no demande les fourchettes.
     *  Précondition : il n'en possède aucune.
     *  Postcondition : quand cette méthode retourne, il possède les deux fourchettes adjacentes à son assiette. */
    public void demanderFourchettes (int no) throws InterruptedException{
        if (no == 0) {
            this.fourchettes[Main.FourchetteDroite(no)].acquire();
            IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.AssietteGauche);
            this.fourchettes[Main.FourchetteGauche(no)].acquire();
            IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.AssietteDroite);
        } else {
            this.fourchettes[Main.FourchetteGauche(no)].acquire();
            IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.AssietteDroite);
            this.fourchettes[Main.FourchetteDroite(no)].acquire();
            IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.AssietteGauche);
        }
    }

    /** Le philosophe no rend les fourchettes.
     *  Précondition : il possède les deux fourchettes adjacentes à son assiette.
     *  Postcondition : il n'en possède aucune. Les fourchettes peuvent être libres ou réattribuées à un autre philosophe. */
    public void libererFourchettes (int no) {
        this.fourchettes[Main.FourchetteDroite(no)].release();
        IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.Table);
        this.fourchettes[Main.FourchetteGauche(no)].release();
        IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.Table);
    }

    /** Nom de cette stratégie (pour la fenêtre d'affichage). */
    public String nom() {
        return "Implantation Sémaphores, stratégie de l'inverse";
    }

}
```

FIGURE 3 – Listing3 : PhiloInv.java

La seconde solution consiste à donner en premier la fourchette droite, puis vérifier si la fourchette gauche est disponible, si elle est disponible le philosophe peut la prendre, dans la cas contraire, le philosophe relache la fourchette de droite, ce qui rend cette solution non bloquante.



```java
// Time-stamp: <08 déc 2009 08:30 queinnec@enseeiht.fr>

import java.util.concurrent.Semaphore;

public class PhiloTestGauche implements StrategiePhilo {

    /** Attributs de classe */
    private static Semaphore[] fourchettes;

    /** Constructeur de classe */
    public PhiloTestGauche (int nbPhilosophes) {
        this.fourchettes = new Semaphore[nbPhilosophes];
        for (int i = 0; i < nbPhilosophes; i++)
            this.fourchettes[i] = new Semaphore(1);
    }

    /** Le philosophe no demande les fourchettes.
     *  Précondition : il n'en possède aucune.
     *  Postcondition : quand cette méthode retourne, il possède les deux fourchettes adjacentes à son assiette. */
    public void demanderFourchettes (int no) throws InterruptedException{
        boolean mange = false;
        while (!mange) {
            this.fourchettes[Main.FourchetteDroite(no)].acquire();
            IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.AssietteGauche);
            if (this.fourchettes[Main.FourchetteGauche(no)].tryAcquire()) {
                IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.AssietteDroite);
                mange = true;
            } else {
                this.fourchettes[Main.FourchetteDroite(no)].release();
                IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.Table);
            }
        }
    }

    /** Le philosophe no rend les fourchettes.
     *  Précondition : il possède les deux fourchettes adjacentes à son assiette.
     *  Postcondition : il n'en possède aucune. Les fourchettes peuvent être libres ou réattribuées à un autre philosophe. */
    public void libererFourchettes (int no) {
        this.fourchettes[Main.FourchetteDroite(no)].release();
        IHMPhilo.poser(Main.FourchetteDroite(no), EtatFourchette.Table);
        this.fourchettes[Main.FourchetteGauche(no)].release();
        IHMPhilo.poser(Main.FourchetteGauche(no), EtatFourchette.Table);
    }

    /** Nom de cette stratégie (pour la fenêtre d'affichage). */
    public String nom() {
        return "Implantation Sémaphores, stratégie test la fourchette gauche";
    }

}
```

FIGURE 4 – Listing4 : PhiloTestGauche.java

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

2.1 Implantation

On peut remarque que dans la version des états, la moitié des philosophes mangent simultanément contrairement aux versions précedente où le parallélisme est limité et à partir d'un certain temps, il y'a q'un seul philosophe qui mange, ainsi que la version des états monopolise les ressources sans avoir de résultats à cause de l'attente active.

2.2 Equité

Un scénario de 4 philosophes, dont deux philosophes face à face sont plus rapides, donc les deux autres philosophes ne mangent pas que si les deux premiers philosophes ne mangent pas, ce qui pourra produire une famine pour les deux autres philosophes.

On peut mettre en place un compteur pour chaque philosophe qui s'incrémente à chaque fois qu'un philosophe mange, le philosophe dont son compteur est le plus petit pourra manger.

Une autre solution possible serait d'avoir une file d'attente des philosophes qui demandent à manger.Ainsi, les philosophes ne pourraient pas bloquer les autres en mangeant constammant s'ils sont plus rapide et chaque philosophe finit par manger.

Dans le pire des cas, il y'a qu'un seul philosophe qui mange contrairement à la première solution où pratiquement la moitié des philosophes mangent au même temps. L'attente maximum pour un philosophe demandeur est d'un tour de table par rapport à un philosophe servis. Cette version a un degré de parallélisme moins performant que la version d'états.