TP3 : Problème des philosophes

Sajid Badr 2SN HPC-BIGDATA

Première approche : les fourchettes sont des ressources critiques :

- **Version de base :** ** PhiloSemBase.java ** et ** PhiloSemBaseInterblocage.java ** :

Pour le cas de base on utilise les sémaphores pour prendre la fourchette droite eu début puis le fourchette gauche.
Et pour visualiser l'interblocage, on ajoute « Thread.sleep(1000); » et on remarque qu'à un moment tous les philosophes ont pris la fourchette droite et attendent celle de gauche ce qui crée un cycle et donc l'interblocage.

Pour remédier à ce problème d'interblocage, on implémente différentes solutions :

- **Version en supprimant une place :** ** PhiloSemPlace.java ** :

Dans cette version on supprime une chaise, donc pour N philosophes on aura que N-1 philosophes qui peuvent manger donc on élimine le cycle et on n'aura plus d'interblocage.

- **Version de fourchette en bloc :** ** PhiloSemBloc.java ** :

Dans cette version, on évite l'interblocage en ajoutant le sémaphore Mutex qui permet à un philosophe de prendre les deux fourchettes à la fois. Mais pour cette version on remarque que si un philosophe attend la fourchette de droite, il bloc tous les autres philosophes de manger.

- **Version Classe Ordonnée :** ** PhiloSemClassesOrdonnees.java ** :

Cette version illustre le principe des classes ordonnées qui est donner à chaque fourchette un numéro tel que chaque philosophe commence par prendre la fourchette d'ordre inférieure puis celle d'ordre supérieur. Ce qui est donné par le faite que tous les philosophes prennent la fourchette de droite (celle d'ordre inférieure) puis la fourchette de gauche (celle d'ordre supérieur) sauf le dernier philosophe qui commence par la fourchette de gauche (ordre "0") puis celle de droite (ordre " N-1").

- **Version en utilisant TryDown :** ** PhiloSemTryDown.java ** :

Cette solution permet à chaque philosophe de prendre sa fourchette de droite, et en essayant de prendre celle de gauche, si elle est déjà prise, il repose la fourchette droite pour ne pas bloquer les autres et il réessaie la même chose jusqu'au moment ou il prend les deux fourchettes.

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

- **Version en contrôler la progression d'un philosophe en fonction de l'état de ses voisins :** ** PhiloSemVoisin.java ** :

Dans cette version, le philosophe decide de manger ou pas dépend de l'état de ses voisins. Si ces voisins sont entrain de penser, il peut alors manger sinon il doit attendre. Et au moment de poser les fourchettes le philosophe teste si ses voisins peuvent manger pour les signaler.

Par rapport à la première approche, on remarque qu'au bout d'un moment tout la moitié des philosophes peut manger. Par contre pour la première approche, on remarque qu'après un moment un seul philosophe qui mange.

Equité :

- Un scénario de 4 philosophes, dont deux qui sont en faces sont plus rapides par rapport au deux autres, peut conduire à la famine dans la cas où ces deux philosophes lents peuvent toujours attendre les deux rapides qui sont pas en parallèles et donc ils pourront plus manger et donc famine.

- Pour éviter ce cas, on peut utiliser une file d'attente dans la quelle on ajoute les philosophes qui demandent et au moment de signaler, on signale le premier philosophe. Comme ça sera sûr que tous les philosophes pourront manger.

- Au pire des cas, on aura un seul philosophe qui mange et les autres soit attendent ou pensent. Par contre la solution optimale permet à la moitié des philosophes de manger.

- Le philosophe peut attendre au maximum un tour de la table si tous les autres demandent il sera donc le dernier à avoir la possibilité de manger.

- On peut remarquer que cette solution revient aux mêmes résultats que celles de la première approche au niveau de degré de parallélisme (au pire des cas un philosophe qui mange) mais qui est plus couteuse au terme de complexité.