Projet Prod Cons

Titouan Larnicol
Thibaut Vegreville

December 2017

1 Introduction

Le but de ce projet était de créer une application utilisant la programmation concurrente avec les Threads dans le langage java où des producteurs et consommateurs communiquent via des messages stocké sur un tampon.
Nous avons réalisé le projet jusqu'à l'objectif 4. Les tests sont bien réalisés via le fichier où l'on a réalisé différentes configuration pour tester plusieurs cas.
Test 1 : On a plus de consommateur que de producteur avec un seul message par producteur.
Test2 : Nombre de producteurs infèrieur au nombre de consommateur
Test3 : Nombre de producteur supèrieur au nombre de Consommateur
Pour l'affichage, nous écrivons simplement quel est le consommateur qui lit le message n X produit par le Producteur P.

2 Objectif 1

On applique la solution directe avec les wait et notify. On doit donc lorsque l'on veut rajouter un message (ou l'enlever) du buffer tester des conditions tel que le buffer ne soit pas de taille nul lorsque l'on veut lire et buffer non plein lorsque l'on veut écrire. Lorsque c'est occupé on utilise la méthode wait puis lorsque la lecture ou l'écriture est fini, on "notifyall" pour réveiller les processus bloqués. Les messages sont numérotés pour pouvoir vérifier si ils sont ensuite bien lus dans l'ordre de création.

3 Objectif 2

Nous avons repris les semaphore de la classe java, et avons implémenté une solution où les consommateur peuvent acceder en même temps que les producteurs. Cependant, deux producteur et deux consommateurs ne peuvent pas agir en même temps (problème de droit d'accée.). Les test sur nos 3 différentes compositions se révélent concluant sur cette v2.

4 Objectif 3

Dans cet objectif, le but était d'insérer les primitives aux bons endroits. J'ai donc fait appels aux primitives dans le get et put de ProdCons. On ajoute le message à l'observateur dans put et on les retire via RetraitMessage dans le get.

5 Objectif 4

La v4 nous aura donné le plus de difficulté car l'algorithme ne nous est pas venu intuitivement. Nous avons donc mis en oeuvre un systéme de copie : un producteur dépose son message dans un nombre d'exemplaire identique au nombre de consomateur. Le buffer de message est en realité un buffer de buffer, dans lequel nous stockons des groupes de message. Les consommateurs prélevent alors sur le premier buffer de x message (x correspondant aux nb de consommateur), et ceux jusqu'a vider le buffer. Le dernier consommateur nettoie la place. Les producteurs dépose leur production sous forme de buffer complet dans le buffer principale, a l'unique condition qu'ils ne soient pas présent dans une lsite de producteur ayant déja déposé et donc la consomation n'a pas été fini. Les consommateur fonctionnent sur le meme principe, ne pouvant re-consommé que si le buffer en cours de consommation est fini (le dernier consommateur relache les autres).

Notre principale déconvenue réside dans le fait que nous n'avons pu remettre les sémaphores dans cette v4, en fonctionnant avec un système de buffers synchrones qui, au final, imite le rôle de sémaphore. Nous n'avons pas eu le temps de refaire la version pour y incorporer les sémaphores Java précédement utiliser.

Nos test se sont révélé concluant pour la v4 également.

6 Objectif 5 6

Nous n'avons pas réalisé ces objectifs.

7 CONCLUSION

Ce projet nous a permis de mettre en application de nombreux points vu en cours Système ce semestre que ce soit en programmation concurente ou sur le langage java simplement. Nous avons eu quelques difficultés sur la mise en place et la compréhension du sujet