Yohan Cottin e1701053 Rosalie Perée e2004724

SYSTEMES D'EXPLOITATION - PROJET

YOHAN COTTIN *e1701053* & ROSALIE PEREE *e2004724*

Dans le cadre du cours de *Système d'exploitation* de la première année de Cyberdéfense à l'ENSIBS, il nous a été demandé d'effectuer un projet sur les sémaphores en Java. Ce document est le rapport portant sur ce projet.

PRODUCTEUR-CONSOMMATEUR

Deux processus se partagent une mémoire tampon de taille fixe (ex un tableau). Un des processus, le producteur, dépose des éléments dans le tampon. Le deuxième processus extrait l'information (le consommateur). Les problèmes surviennent lorsque :

- *Le tampon est plein le producteur ne peut plus déposer d'information.*
- *Le tampon est vide le consommateur ne peut plus extraire de l'information*

SANS SEMAPHORE

Sans sémaphores, le programme fonctionne correctement sauf dans deux cas : si le tampon est plein et qu'on tente d'y déposer de l'information, ou si le tampon est vide et qu'on tente de retirer de l'information.

```
Version sans Semaphores
0
Pas d'article
1
Pas d'article
2
3
4
Pas d'article
5
Pas d'article
6
Pas d'article
```

S'il y a plusieurs producteurs ou consommateurs, ils peuvent essayer d'écrire simultanément dans le tampon, ce qui peut donner à des comportements imprévisibles.

Le sémaphore permet de régler ces problèmes.

Les chiffres sur l'image ci-contre représentent le nombre d'articles total consommés, la phrase "pas d'article" apparaissant lorsque le consommateur tente d'accéder à un tampon vide.

AVEC SEMAPHORES

Pour pallier les problèmes soulevés dans le point précédent, on peut utiliser des sémaphores. On veut vérifier deux conditions :

- Si le tampon est vide, le processus consommateur doit être bloqué jusqu'à ce qu'un producteur dépose un élément.
- L'accès aux cases du tampon doit être protégé. Le consommateur ne peut extraire l'élément que le producteur est en train de déposer.

Dans le constructeur de la classe nous trouvons les éléments suivants :

```
public ProducteurConsommateur(){
    super();
    mutex=new Semaphore( permits: 1);
    places=new Semaphore(MAX);
    articles=new Semaphore( permits: 0);
    tampon=new int[MAX];
}
```

- *places* permet de bloquer le producteur quand il n'y a plus de place dans le tampon
- *articles* permet de bloquer le consommateur quand y a pas d'article dans le tampon
- *mutex* permet de s'assurer qu'il n'y a pas d'accès simultané au tampon.

```
Version avec Semaphores
0
1
2
3
4
5
6
7
8
9
```

Les chiffres sur l'image ci-contre représentent le nombre d'articles total consommés, la phrase "pas d'article" n'apparaissant plus car le consommateur ne peut pas accéder à un tampon vide.

MESSAGES PUBLICITAIRES

Le modèle lecteur(s)-rédacteur : Les lecteurs peuvent consulter l'information qu'un rédacteur construit. Les contraintes sont les suivantes :

- *Il n'y a pas de limite sur le nombre de lecteurs en parallèle.*
- *Les lectures s'effectuent en exclusion mutuelle avec le rédacteur.*

VERSION 1 : PRIORITE DES LECTEURS SUR LE REDACTEUR

Nous disposons d'un *MessageBoard* qui est simplement l'équivalent d'un panneau publicitaire.

Nos lecteurs vont bloquer en écriture le panneau le temps de lire les messages, et de les imprimer dans la console. Pendant ce temps, aucune écriture n'est possible. On peut avoir plusieurs lecteurs simultanément ou à la suite tant que le sémaphore n'a pas été libéré. Le panneau est bloqué à partir du premier lecteur et débloqué quand plus aucun lecteur ne le lit.

```
Message num 0 : 0
Message num 1 : 0
Message num 2 : 0
Message num 3 : 0
Message num 4 : 0
Message num 0 : 1
Message num 1 : 1
Message num 2 : 1
Message num 3 : 1
Message num 4 : 1
Message num 0 : 2
Message num 1 : 2
```

Le rédacteur va écrire sur le panneau en boucle. Il bloque, il écrit cinq messages et il débloque. Les lecteurs peuvent ensuite prendre la suite si besoin. Sinon, le rédacteur continue à bloquer, écrire et puis débloquer.

Le délai entre les affichages des logs dans la console vient du fait que des lecteurs accèdent au panneau et qu'aucune écriture n'est donc possible. Si on met trop de lecteurs ils vont bloquer le rédacteur.

On peut voir sur la capture d'écran que rien n'interrompt l'écriture lors de l'affichage. On le verra surtout en termes de temps entre les affichages, ce qu'on ne peut pas voir avec une capture.

VERSION 2 : PRIORITE DU REDACTEUR SUR LES LECTEURS

Avec une priorité des rédacteurs sur les lecteurs, il y a des changements par rapport à la section précédente.

Pour la priorité du rédacteur, on utilise désormais un sémaphore de plus. On a également 4 variables d'entiers qui représentent les lecteurs et les rédacteurs, soit actifs, soit en file d'attente. On n'a de lecteurs en attente que si un rédacteur est en train d'écrire, mais si on a déjà quelqu'un en lecture ou en écriture un rédacteur sera mis en attente.

Le lecteur va bloque l'accès aux variables et va regarder s'il y a un rédacteur en écriture ou des rédacteurs en attente. Si oui, il marque qu'il y a un lecteur de plus en attente et libère les variables. On va ensuite bloquer le lecteur pour attendre que le rédacteur ait fini d'écrire. Une fois que le lecteur est débloqué il reprend l'accès sur les variables et indique qu'il y a un lecteur de moins en attente. Il marque qu'il y a un lecteur actif de plus sur le panneau d'affichage. Il va ensuite libérer la variable et lire le panneau d'affichage. Une fois qu'il a fini de lire il rebloque les variables et marque qu'il y a un lecteur actif de moins. S'il n'y a plus de lecteur actif et qu'au moins un rédacteur est en attente alors on libère un rédacteur. On débloque ensuite les variables.

Le rédacteur commence par bloquer les variables pour faire des modifications sur les 4 entiers. S'il y a un lecteur ou un rédacteur actif, il libère les variables et se met en attente, il est bloqué. Une fois qu'il est débloqué il reprend la main sur les variables, il y a un rédacteur en attente de moins ; il y a dons un rédacteur actif de plus. Il libère les variables et écrit ensuite sur le panneau d'affichage avant de bloquer à nouveau les variables. Il y a un rédacteur actif de moins. S'il y a au moins un rédacteur en attente, il libère un rédacteur. Sinon, s'il y a au moins un lecteur en attente, il libère tous les lecteurs. Il libère ensuite les variables

```
Message num 0 : 0
Message num 1 : 0
Message num 2 : 0
Message num 3 : 0
Message num 4 : 0
Message num 0 : 1
Message num 1 : 1
Message num 2 : 1
Message num 3 : 1
Message num 4 : 1
Message num 0 : 2
Message num 1 : 2
Message num 2 : 2
```

Si on met trop de rédacteurs ils vont bloquer les lecteurs.

On peut voir sur la capture d'écran ci-contre que rien n'interrompt l'écriture au cours de l'affichage. On verra surtout la bonne utilisation des sémaphores en termes de temps, ce qui n'est pas visible sur une capture d'écran.

CONCLUSION

Lors de la réalisation de ce projet, nous n'avons rencontré qu'un problème principal avec un *timer* lancé au mauvais endroit dans les exercices avec le producteur consommateur, ce qui causait la récupération du mutex avant celui qui était en attente ne puisse le récupérer. A part cela, nous n'avons pas rencontré de problème particulier.