TD5 : Synchronisation de threads

1 Concurrence et atomicité

On suppose une variable globale X initialisée à zéro et partagée entre N threads «incrémenteurs» et N threads «décrémenteurs». Chaque thread incrémenteur exécute, de façon non atomique,[1] l'opération `X=X+2`, puis se termine. De même, chaque décrémenteur fait `X=X-3`.
On lance les 2N threads en concurrence et on attend la fin de l'exécution.

Exercice 1 Quelle est la valeur maximale possible pour la variable X ? la valeur minimale ? Donnez une trace d'exécution justifiant chaque réponse.

Exercice 2 Quel est l'ensemble des valeurs possibles pour X ?

Exercice 3 Même question, mais en supposant maintenant que l'exécution de chaque thread est atomique.

2 Sections critiques et interblocages

On s'intéresse au programme ci-dessous, où deux ressources critiques X et Y sont partagées par quatre threads concurrents A à D. Chaque ressource R est protégée par un verrou pour garantir son accès exclusif : un thread fait toujours `lock(LR)` avant d'utiliser R et `unlock(LR)` ensuite.

Variables partagées

```
1 mutex LX, LY;
```

Thread A

```
1 while(true) {
2   lock(LX);
3   A3; // A utilise X
4   unlock(LX);
5 }
```

Thread B

```
1 while(true) {
2   lock(LY);
3   B3; // B utilise Y
4   unlock(LY);
5 }
```

Thread C

```
1 while(true) {
2   lock(LX);
3   C3; // C utilise X
4   lock(LY);
5   C5; // C utilise X et Y
6   unlock(LX);
7   unlock(LY);
8 }
```

Thread D

```
1 while(true) {
2   lock(LY);
3   D3; // D utilise Y
4   lock(LX);
5   D5; // D utilise X et Y
6   unlock(LX);
7   unlock(LY);
8 }
```

Exercice 4 Donnez un scénario d'exécution menant à un interblocage. Exprimez votre réponse sous la forme d'une trace d'exécution de la forme (A1, B1, A2...) pour dire «A exécute sa ligne n° 1, puis B exécute sa ligne n° 1, puis A exécute sa ligne n° 2...»

Exercice 5 Donnez un scénario dans lequel toutes les sections critiques (A3, B3, D5, etc) sont exécutées, sans pour autant provoquer d'interblocage.

Exercice 6 (facultatif) Réécrivez les synchronisations de ce programme de façon à garantir l'absence d'interblocage. Vous devez bien sûr garantir l'exclusion mutuelle sur chaque ressource X et Y. Par contre, on ne veut pas limiter artificiellement le parallélisme : par exemple, A3 et B3 doivent pouvoir s'exécuter simultanément, de même que A3 et D3, ou encore C3 et D3, etc.

1. autrement dit, on considère que la lecture, le calcul, et l'écriture sont trois actions distinctes, qui peuvent arriver de façon concurrente vis-à-vis de celles autres threads

3 Variantes sur le producteur-consommateur

On s'intéresse à une version du producteur-consommateur similaire à celle vue en cours mais utilisant une seule variable de condition, comme illustré ci-dessous.

Variables partagées

```
item_t buffer[N];
int count=0;
mutex L;
cond_var bell;
```

Producteur

```
1  int in = 0;
2  while(1)
3  {
4     item=produce()
5     lock(L);
6     while(count == N)
7     {
8        wait(bell, L);
9     }
10    buffer[in] = item;
11    in = (in+1) % N;
12    count = count + 1;
13    signal(bell);
14    unlock(L);
15 }
```

Consommateur

```
1  int out = 0;
2  while(1)
3  {
4     lock(L);
5     while(count == 0)
6     {
7        wait(bell, L);
8     }
9     item = buffer[out];
10    out = (out+1) % N;
11    count = count - 1;
12    signal(bell);
13    unlock(L);
14    consume(item);
15 }
```

Exercice 7 Est-ce que cette implémentation est toujours correcte, ou bien a-t-on introduit des problèmes de concurrence (data race, interblocage...) ? En fonction de votre réponse, justifiez votre avis par une phrase rédigée, ou donnez un exemple d'exécution menant à un problème.

Exercice 8 Même question, mais en remplaçant aussi les appels `signal()` par des appels `broadcast()`.

4 Signalisation et précédence

Exercice 9 Soit le programme ci-dessous, composé de deux threads concurrents. Complétez ce programme pour qu'il respecte le graphe de précédence A() → B(), c'est à dire pour qu'il garantisse que l'exécution de A() soit terminée avant le le début de B(). Vous pouvez utiliser des verrous, des variables de condition, et des variables partagées.

Conditions initiales

Thread A

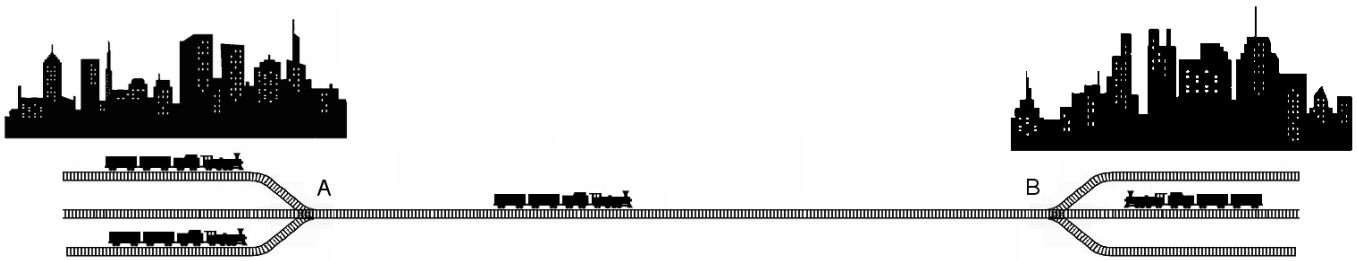
```
A();
```

Thread B

```
B();
```

5 Moniteurs

Exercice 10 Deux villes A et B sont reliées par une ligne de chemin de fer à une seule voie. Plusieurs trains peuvent circuler dans le même sens, de A vers B, ou de B vers A, mais des trains circulant en sens opposés ne doivent pas occuper la voie en même temps.



On modélise cette situation sous la forme d'un programme concurrent, dans lequel une quantité arbitraire de threads exécutent chacun l'un ou l'autre des comportements ci-dessous :

Trains de A à B

```
1 // demander le droit de passage A vers B
2 circuler de A vers B
3 // sortir de la voie en B
```

Trains de B à A

```
1 // demander le droit de passage B vers A
2 circuler de B vers A
3 // sortir de la voie en A
```

Implémentez, pour chaque type de train, la procédure de demande d'autorisation et celle de sortie de voie. Vous pouvez utiliser des verrous, des variables de condition, et des variables partagées.