Mutex et sémaphore

Quentin Fortier

January 29, 2025

Section critique

Définition

Une section critique est un bloc de code qui vérifie :

- Exclusion mutuelle : il ne peut y avoir qu'un seul thread à la fois dans la section critique.
- Absence de famine : un thread ne doit pas attendre indéfiniment pour entrer dans la section critique.

Section critique

Définition

Une section critique est un bloc de code qui vérifie :

- Exclusion mutuelle : il ne peut y avoir qu'un seul thread à la fois dans la section critique.
- Absence de famine : un thread ne doit pas attendre indéfiniment pour entrer dans la section critique.

Il est préférable d'utiliser le moins possible de sections critiques, car elles limitent la parallélisation du programme.

Mutex

Définition

Un mutex (*mutual exclusion*) ou verrou est un objet ayant trois opérations :

- Création du mutex.
- Verrouillage (*lock*) du mutex.
- Déverrouillage (*unlock*) du mutex.

Tel que :

- Au plus un thread peut verrouiller le mutex à la fois.
- Le bloc de code entre le verrouillage et le déverrouillage est une section critique.

Mutex

```c
int counter;
pthread_mutex_t mutex;

void *increment(void *arg){
    for (int i = 1; i <= 1000000; i++) {
        pthread_mutex_lock(&mutex);
        counter++;
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Le résultat est toujours 2000000 (avec 2 threads).

Mutex

Question

Comment implémenter un mutex ?

Pour simplifier, on va implémenter un mutex pour deux threads seulement (alors que les mutex en C ou OCaml fonctionnent avec un nombre arbitraire de threads).
On suppose de plus que les threads sont numérotés 0 et 1.

Mutex : Tentative d'implémentation 1

On essaie d'utiliser un tableau de booléens `m` tel que `m[i]` détermine si le thread `i` est dans la section critique.

Question

L'implémentation suivante de mutex garantie t-elle exclusion mutuelle et/ou absence de famine ?

lock(i)

Tant que `m[1 - i]` :
  Attendre
`m[i] = true`

create()

`m = [false, false]`

unlock(i)

`m[i] = false`

Mutex : Tentative d'implémentation 1

- L'absence de famine est vérifiée

Mutex : Tentative d'implémentation 1

- L'absence de famine est vérifiée : si le thread `i` est bloqué dans le `Tant que` alors `m[i]` est `false` donc l'autre thread peut entrer dans la section critique.
- L'exclusion mutuelle n'est pas garantie

Mutex : Tentative d'implémentation 1

- L'absence de famine est vérifiée : si le thread `i` est bloqué dans le `Tant que` alors `m[i]` est `false` donc l'autre thread peut entrer dans la section critique.
- L'exclusion mutuelle n'est pas garantie : si le thread 0 sort du `Tant que` et le thread 1 sorte de `Tant que` avant que le thread 0 n'ait pu mettre `m[0]` à `true`.

Mutex : Tentative d'implémentation 2

Question

L'implémentation suivante de mutex garantie t-elle exclusion mutuelle et/ou absence de famine ?

lock(i)

```
m[i] = true
Tant que m[1 - i] :
    Attendre
```

create()

```
m = [false, false]
```

unlock(i)

```
m[i] = false
```

Mutex : Tentative d'implémentation 2

- L'absence de famine n'est pas garantie

Mutex : Tentative d'implémentation 2

- L'absence de famine n'est pas garantie : si les deux threads exécutent `m[i] = true` avant de rentrer dans le `Tant que`.
- L'exclusion mutuelle est vérifiée

Mutex : Tentative d'implémentation 2

- L'absence de famine n'est pas garantie : si les deux threads exécutent `m[i] = true` avant de rentrer dans le `Tant que`.
- L'exclusion mutuelle est vérifiée : supposons par l'absurde que les deux threads soit dans la section critique en même temps. Supposons que le thread 0 est entré en premier. Alors, à ce moment, `m[1]` est `false` donc le thread 1 n'a pas encore exécuté `m[1] = true`. Donc le thread 1 ne peut pas entrer dans la section critique.

Mutex : Tentative d'implémentation 3

Question

L'implémentation suivante de mutex garantie t-elle exclusion mutuelle et absence de famine ?

`lock(i)`

Tant que `tour = 1 - i` :
  Attendre

`create()`

```
tour = 0
```

`unlock(i)`

```
tour = 1 - i
```

Mutex : Tentative d'implémentation 4

On combine les deux tentatives précédentes.

Question

L'implémentation suivante de mutex garantie t-elle exclusion mutuelle et absence de famine ?

lock(i)

```
attente[i] = true
tour = i
Tant que tour = 1 - i et attente[1 - i] :
    Attendre
```

create()

```
attente = [false, false]
tour = 0
```

unlock(i)

```
tour = 1 - i
attente[i] = false
```

Mutex : Algorithme de Peterson

L'algorithme de Peterson permet d'implémenter un mutex.

lock(i)

```
attente[i] = true
tour = 1 - i
Tant que tour = 1 - i et attente[1 - i] :
    Attendre
```

create()

```
attente = [false, false]
tour = 0
```

unlock(i)

```
attente[i] = false
```

Mutex : Algorithme de Peterson

Théorème