Programmation parallèle sans verrous

Félix-Antoine Ouellet

Université de Sherbrooke

16 octobre 2014

Plan

Programmation parallèle traditionnelle

Exemple

```
void ThreadSafeIntStack::push(int val) {
  std::lock_guard<std::mutex> lock(m_Mutex);
  m_Stack.push(val);
}

int ThreadSafeIntStack::pop() {
  std::lock_guard<std::mutex> lock(m_Mutex);
  m_Stack.pop();
}
```

Problème
Coût des verrous

Traditionnellement:

1. Attendre que le mutex soit disponible
2. Acquérir le mutex
3. Effectuer opération(s)
4. Libérer le mutex

Plan

Définitions

Il existe diverses définitions selon le niveau de garanties fourni.

- Sans attente
- Sans verrous
- Sans obstruction

Sans attente

Chaque fil d'exécution s'exécute dans un nombre fini d'étapes sans égard pour des facteurs externes.

Sans attente

Chaque fil d'exécution s'exécute dans un nombre fini d'étapes sans égard pour des facteurs externes.
Exemple:

```
void IncrementRefCounter(Object *obj) {
    atomic_increment(obj->rc);
}
```

Sans verrous

Le système en entier va continuer de progresser malgré que certains fils d'exécution ne progressent pas.

Sans verrous

Le système en entier va continuer de progresser malgré que certains fils d'exécution ne progressent pas.
Exemple:

```
void StackPush(Stack *s, Node *n) {
  Node* head;
  do {
    head = s->head;
    n->next = head;
  }
  while (!CompareExchange(s->head, head, n));
}
```

Sans obstruction

Un fil d'exécution exécuter en isolation va terminer dans un nombre fini d'étapes.

Sans obstruction

Un fil d'exécution exécuter en isolation va terminer dans un nombre fini d'étapes.
Exemple:

?

Plan

Atomiques?

Test 1

```
uint64_t sharedValue = 0;

/* . . . */

sharedValue = 0x100000002;
```

Atomiques?
Test 1

Compilé pour x86:

```
/* . . . */

mov DWORD PTR sharedValue, 2
mov DWORD PTR sharedValue+4, 1

/* . . . */
```

Atomiques?

Test 2

```
strd r0, r1, [r2]
```

Read-Modify-Write

- Classe d'opérations atomiques
- Souvent utilisé pour implémenter des mutex
- Exécute une lecture et une écriture

Compare-And-Swap

Haut niveau

```
int CompareAndSwap(int *reg, int oldVal,
                   int newVal) {
  int r = *reg;
  if (*reg == oldVal)
      *reg = newVal;
  return r;
}
```

Compare-And-Swap
Assembleur

CMPXCHG

Compare-And-Swap

Pratique

```
void LockFreeList::PushFront(Node *newHead) {
  for (;;) {
    Node *oldHead = m_Head;
    newHead->next = oldHead;
    if (CompareAndSwap(&m_Head, newHead,
                       oldHead) == oldHead)
      return;
  }
}
```

Plan

Problème ABA
Description sommaire

Modifications par un fil d'exécution à une structure partagée dont les autres fils d'exécution n'ont pas connaissance.

Problème ABA

Exemple - Code

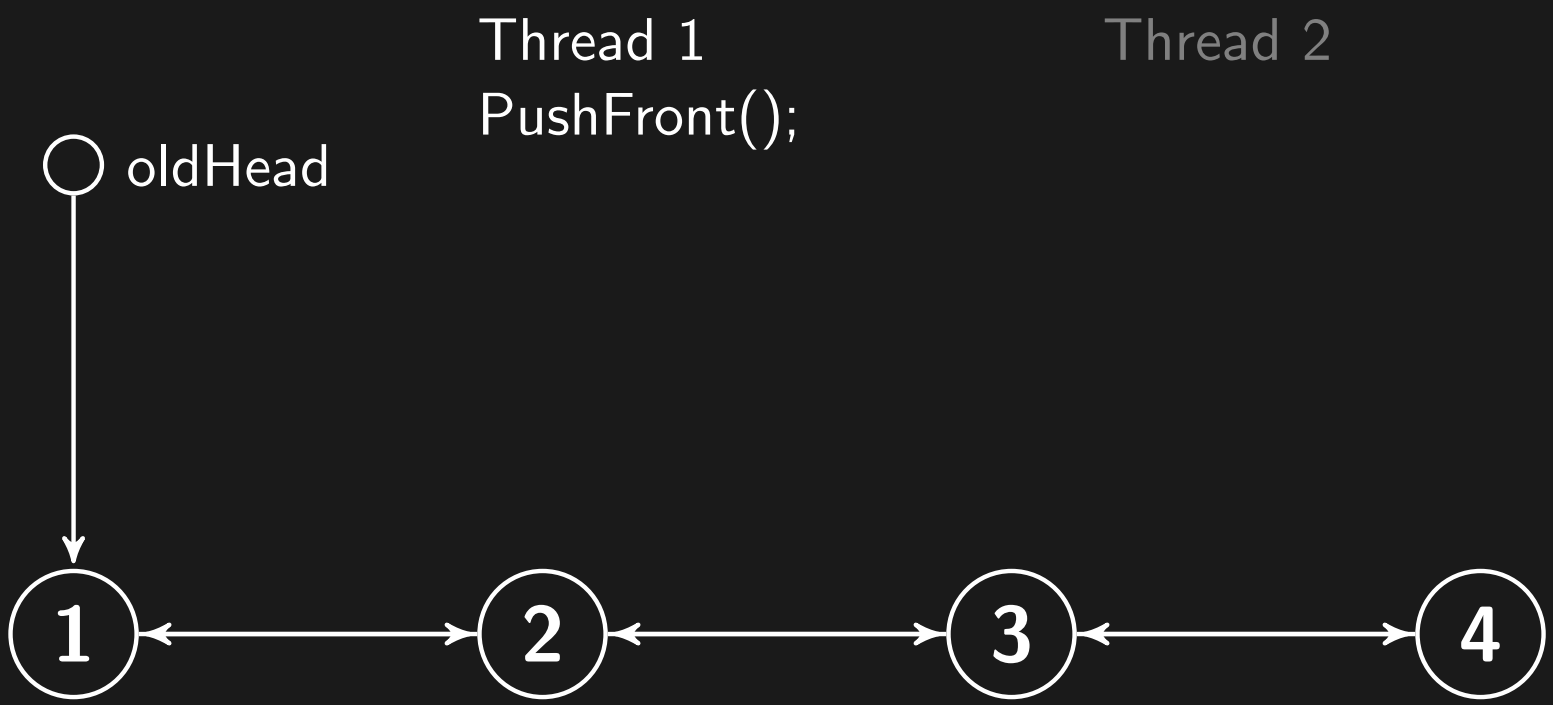
```
void LockFreeList::PushFront(Node *newHead) {
  for (;;) {
    Node *oldHead = m_Head;
    newHead->next = oldHead;
    if (CompareAndSwap(&m_Head, newHead,
                       oldHead) == oldHead)
      return;
  }
}
```

Problème ABA

Exemple - Exécution

Problème ABA

Exemple - Exécution



Problème ABA

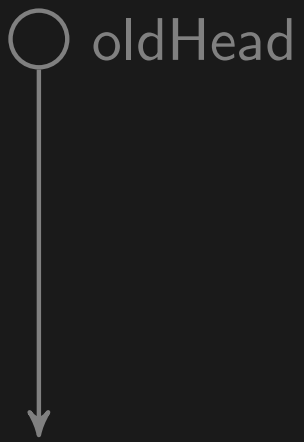
Exemple - Exécution

Thread 1
PushFront();

Thread 2
PopFront();

Problème ABA

Exemple - Exécution



Thread 1
PushFront();

Thread 2
PopFront();
PopFront();

Problème ABA

Exemple - Exécution

Thread 1
PushFront();

Thread 2
PopFront();
PopFront();
PushFront(node1);

Problème ABA
Solutions potentielles

- Tags
- Réclamation différée
- Acteurs

Réordonnancement mémoire

Au départ: $X = 0$, $Y = 0$

Réordonnancement mémoire

Au départ: $X = 0$, $Y = 0$

Thread 1	Thread 2
$X = 1$	$Y = 1$
$x = Y$	$y = X$

Réordonnancement mémoire

Au départ: $X = 0$, $Y = 0$

Thread 1	Thread 2
$X = 1$	$Y = 1$
$x = Y$	$y = X$

À la fin: $\{x = 1, y = 0\}$, $\{x = 0, y = 1\}$, $\{x = 1, y = 1\}$, $\{x = 0, y = 0\}$

Modèle mémoire
Général

- Existe au niveau du langage et du processeur
- Décrit l'interaction possible entre fils d'exécution
  - Atomicité
  - Visibilité
  - Ordonnancement

Modèle mémoire

Langages

- Cohérence séquentielle sur les variables atomiques
  - On impose un ordre global sur les accès mémoire pour préserver l'ordre du programme
  - Utilisation de barrières mémoire en dessous des couvertures
  - Exemple: *volatile* en Java, std::atomic<T>

Modèle mémoire
Application

Au départ: $X = 0$, $Y = 0$ (variables atomiques)

Thread 1	Thread 2
$X = 1$	$Y = 1$
$x = Y$	$y = X$

À la fin: $\{x = 1, y = 0\}$, $\{x = 0, y = 1\}$, $\{x = 1, y = 1\}$

Plan

Conclusion

- On jongle avec des lames de rasoir

Conclusion

- On jongle avec des lames de rasoir
- Pas nécessairement généralisable