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();
}
```

Problèmes

- Coût des verrous
- Inversion de priorités

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

Read-Modify-Write

Haut niveau

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

Read-Modify-Write

Opérations

- Test-And-Set
- Fetch-And-Add
- Compare-And-Swap

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;
  }
}
```

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

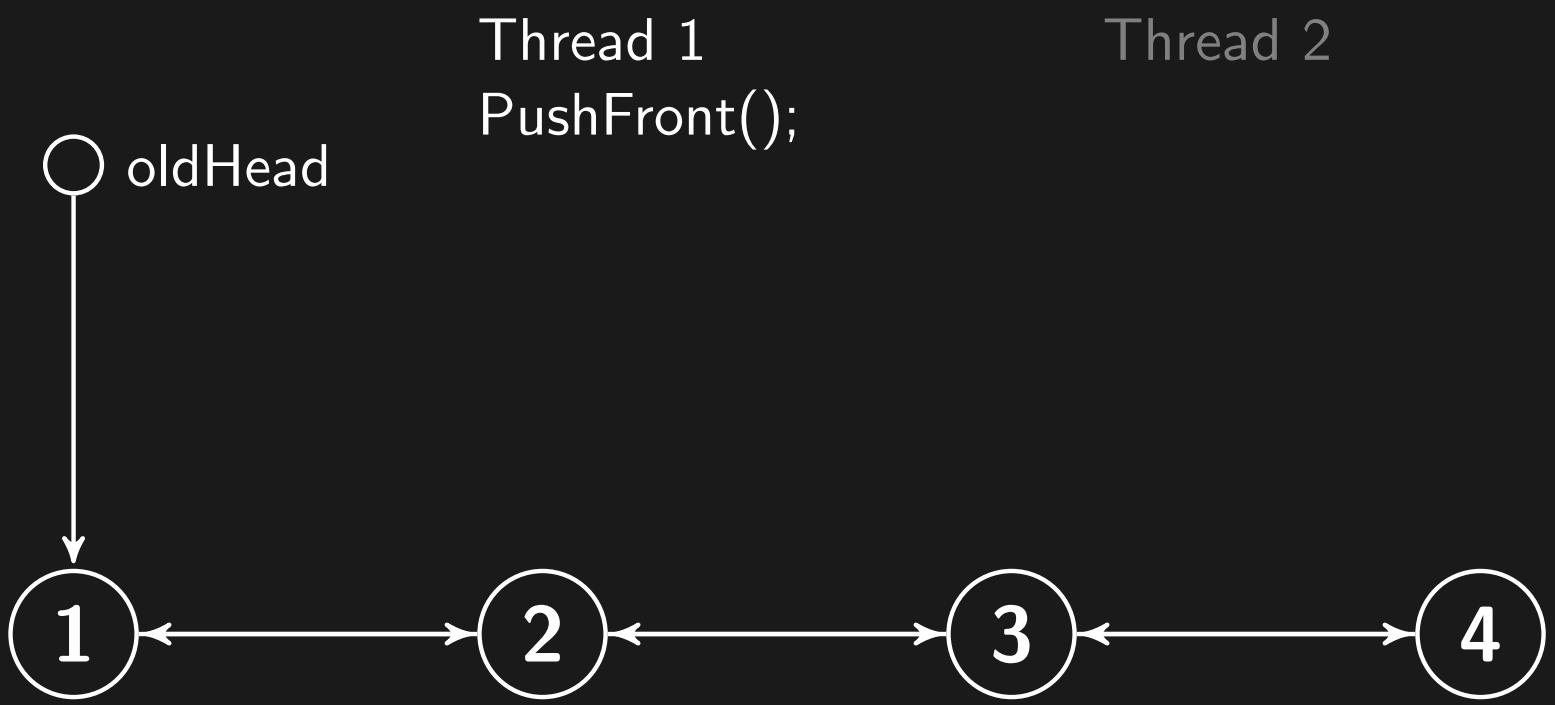
```
void LockFreeList::PushFront(Node *newHead) {
  for (;;) {
    Node *oldHead = m_Head;
    newHead->next = oldHead;
    if (CompareAndSwap(&m_Head, newHead,
                       oldHead) == oldHead)
      return;
  }
}
```

Problème ABA

Exemple - Exécution

Problème ABA

Exemple - Exécution



Problème ABA

Exemple - Exécution

Thread 1
PushFront();

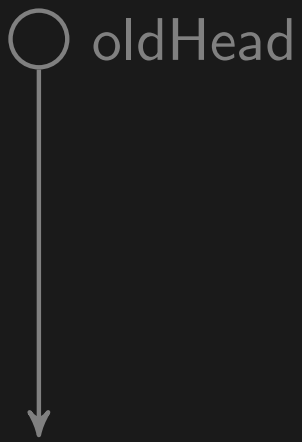
Thread 2
PopFront();

Problème ABA

Exemple - Exécution

Thread 1
PushFront();

Thread 2
PopFront();
PopFront();



Problème ABA

Exemple - Exécution

Thread 1
PushFront();

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

Plan

Modèle mémoire

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

Visibilité

Problème

```
// Processeur #1
while (f) {
  print x;
}

// Processeur #2
x = 42;
f = false;
```

Visibilité
Solution

```
// Processeur #1
while (f);
WriteMemoryBarrier();
print x;

// Processeur #2
x = 42;
WriteMemoryBarrier();
f = false;
```

Ordonnancement

Problème

```
struct foo {
  int a;
  int b;
};
struct foo *gp = NULL;
/* . . . */
p = kmalloc(sizeof(*p), GFP_KERNEL);
p->a = 1;
p->b = 2;
gp = p;
```

Ordonnancement

Solution

```
struct foo {
  int a;
  int b;
};
struct foo *gp = NULL;
/* . . . */
p = kmalloc(sizeof(*p), GFP_KERNEL);
WriteMemoryBarrier();
p->a = 1;
p->b = 2;
gp = p;
WriteMemoryBarrier();
```

Plan

Conclusion

- On jongle avec des lames de rasoir

Conclusion

- On jongle avec des lames de rasoir
- Il faut connaître ses outils logiciels et matériels