Introduction to Lock-Free Programming

Olivier Goffart

2014

About Me

- QStyleSheetStyle
- Itemviews
- Animation Framework
- QtScript (porting to JSC and V8)
- QObject, moc
- QML Debugger
- Modularisation
- ...

About Me

Offering Qt help and services: Visit `http://woboq.com`

C++ Code browser: `http://code.woboq.org`

Goal of this presentation

- Introduction to Lock-Free programming

Singleton

```
1  class MySingleton {
2      static MySingleton *s_instance;
3      static QMutex s_mutex;
4
5  public:
6
7      static MySingleton *instance()
8      {
9          QMutexLocker lock(&s_mutex);
10         if (!s_instance) {
11             s_instance = new MySingleton();
12         }
13         return s_instance;
14     }
15
16     // ....
17 };
```

Singleton (wrong)

```
1   static MySingleton *instance()
2   {
3       if (!s_instance) {
4           QMutexLocker lock(&s_mutex);
5           if (!s_instance) {
6               s_instance = new MySingleton();
7           }
8       }
9       return s_instance;
10  }
```

Computer architecture

- Compiler re-order
- CPU out of order execution
- Caches, Write buffers

Singleton (Better)

```
1  class MySingleton {
2      static QAtomicPointer<MySingleton> s_instance;
3      static QMutex s_mutex;
4  public:
5      static MySingleton *instance()
6      {
7          MySingleton *inst = s_instance.loadAcquire();
8          if (!inst) {
9              QMutexLocker lck(&s_mutex);
10             if (!s_instance.load()) { // relaxed
11                 inst = new MySingleton();
12                 s_instance.storeRelease(inst);
13             }
14         }
15         return inst;
16     }
17 };
```

Singleton (Best)

```
1       static MySingleton *instance()
2       {
3           static MySingleton inst;
4           return &inst;
5       }
```

Singleton (Best)

```
1       static MySingleton *instance()
2       {
3           static MySingleton inst;
4           return &inst;
5       }
```

See also: `Q_GLOBAL_STATIC`

C++11 Memory model

C++98

No mentions of threads.

The compiler is allowed to do any optimisation that is consistant to a single thread.

C++11 Memory model

C++98

No mentions of threads.

The compiler is allowed to do any optimisation that is consistant to a single thread.

C++11

- Defines race condition
- Restricts what kind of optimisation the compiler is allowed to do in regards to threading.
- `std::atomic` , `std::thread`, `std::mutex`

C++11 Memory model

C++11 §1.10

> *21. The execution of a program contains a data race if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.*

Lock-Free programming

What's wrong with mutexes?

What's wrong with mutexes?

- All threads have to wait if a thread holding a lock is descheduled.
- More context switches waste CPU time.
- For real-time applications: priority inversion, unsafe in interrupts handlers, convoying.

Lock-free algorithms

- Sometimes faster
- No risks of deadlock, even if a thread is terminated/killed
- More difficult to design and understand

Lock-free algorithms

- Sometimes faster
- No risks of deadlock, even if a thread is terminated/killed
- More difficult to design and understand, but also fun

QAtomicInt/QAtomicPointer

API

- testAndSet
- fetchAndStore
- fetchAndAdd

Memory Ordering

- Ordered
- Acquire
- Release
- Relaxed

Mix and Match

```
1 bool QAtomicInt::testAndSetAcquire(int expectedValue,
2                                    int newValue)
3 int QAtomicInt::fetchAndAddOrdered(int valueToAdd)
4 T *QAtomicPointer<T>::fetchAndStoreRelaxed(T *newValue)
```

Fetch and Store

```
1  T *QAtomicPointer<T>::fetchAndStore...(T *newValue)
2  {
3    T *oldValue = _q_value;
4    _q_value = newValue;
5    return oldValue;
6  }
```

Fetch and Add

```
1  int QAtomicInt::fetchAndAdd...(int valueToAdd)
2  {
3    int oldValue = _q_value;
4    _q_value += valueToAdd;
5    return oldValue;
6  }
```

Test and Set

```
1  bool QAtomicInt::testAndSet...(int expectedValue,
2                                 int newValue)
3  {
4    if (_q_value != expectedValue)
5      return false;
6    _q_value = newValue;
7    return true;
8  }
```

Memory ordering

Acquire

Memory access following the atomic operation may not be re-ordered before that operation.

Release

Memory access before the atomic operation may not be re-ordered after that operation.

Ordered

Same `Acquire` and `Release` combined: operations may not be re-ordered

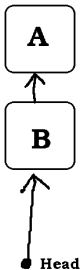
Relaxed

Operations may be re-ordered before or after.

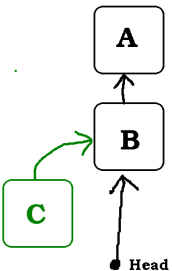
Singleton (Lock-free)

```
1  class MySingleton {
2      static QAtomicPointer<MySingleton> s_instance;
3
4  public:
5      static MySingleton *instance()
6      {
7          MySingleton *inst = s_instance.loadAcquire();
8          if (!inst) {
9              inst = new MySingleton();
10             if (!s_instance.testAndSetRelease(0, inst)) {
11                 delete inst;
12                 inst = s_instance.loadAcquire();
13             }
14         }
15         return inst;
16     }
17 };
```

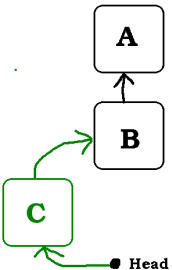
Lock-Free Stack



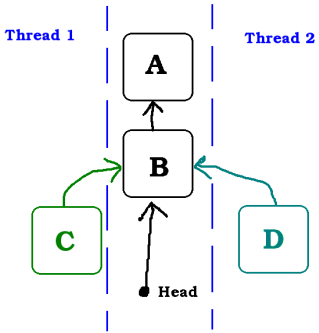
Lock-Free Stack (Push)



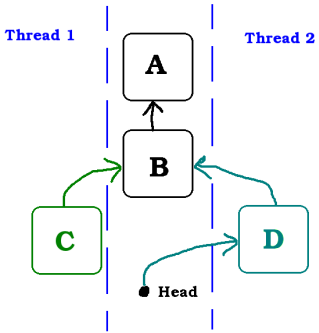
Lock-Free Stack (Push)



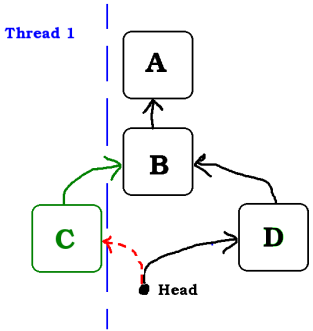
Lock-Free Stack (Push)



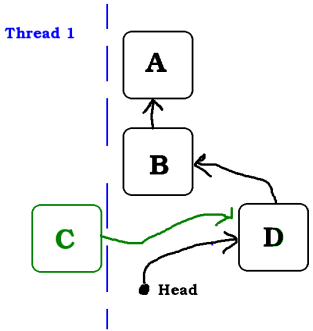
Lock-Free Stack (Push)



Lock-Free Stack (Push)



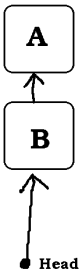
Lock-Free Stack (Push)



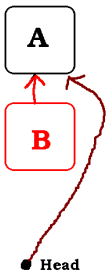
Lock-free stack

```
1  struct Stack {
2      QAtomicPointer<Node> head;
3      void push(Node *n) {
4          do {
5              n->next = head.loadAcquire();
6          } while(!head.testAndSetOrdered(n->next, n));
7      }
8      // ...
9  };
```

Lock-Free Stack (Pop)



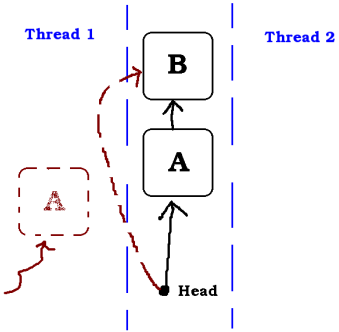
Lock-Free Stack (Pop)

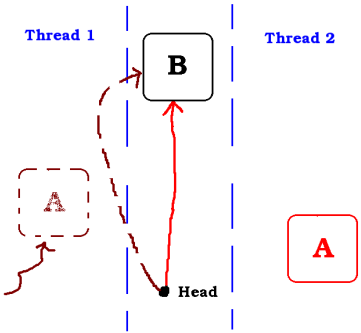


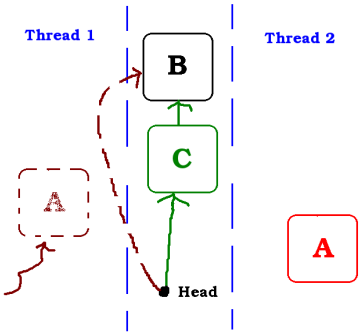
Lock-free stack Pop (wrong)

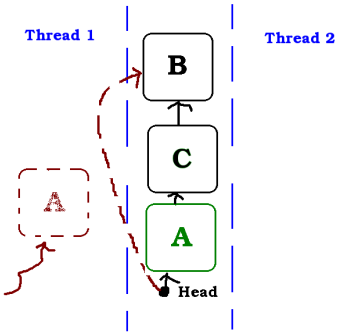
```
1  struct Stack {
2      QAtomicPointer<Node> head;
3      // ...
4      Node *pop() {
5          Node *n;
6          do {
7              n = head.loadAcquire();
8          } while(n && !head.testAndSetOrdered(n, n->next));
9          return n;
10     }
11 };
```

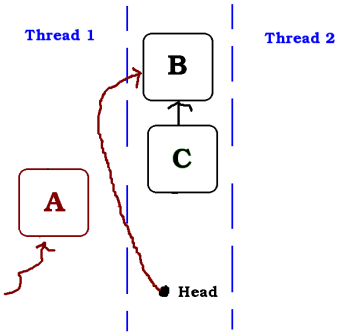
ABA Problem

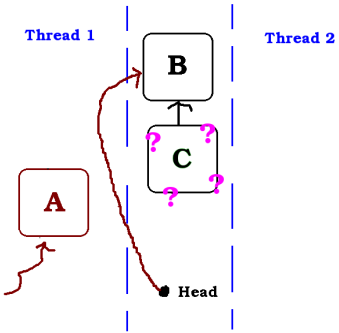



Thread 1
Thread 2
B
A
A
Head


Thread 1
Thread 2
B
C
A
A
Head


Thread 1
Thread 2
B
C
A
A
Head


Thread 1
Thread 2
B
C
A
Head


Thread 1
Thread 2
B
C
A
Head

ABA Problem

Solutions

ABA Problem

Solutions

- Add a serial number
- Multiple words compare and swap
- Garbage collector / Reference count
- Hazard pointers

Example in Qt

- Reference counting
- `QMutex`
- `Q_GLOBAL_STATIC`
- Allocation of timer ids
- ...

Other examples

- RCU (Read-copy-update)
- Multiple words compare and swap.
- Transactional memory

Transactional memory

In the Future... (N3718) :

```
1   void push(Node *n) {
2       transaction_atomic {
3           n->next = head;
4           head = n;
5       }
6   }
7
8   Node *pop() {
9       Node *n;
10      transaction_atomic {
11          n = head;
12          if (n)
13              head = n->next;
14      }
15      return n;
16  }
```

Conclusion

- Use mutexes.
- Profile.

The END

Questions

The END

Questions

olivier@woboq.com

woboq

Visit http://woboq.com.

Read More: http://woboq.com/blog/introduction-to-lockfree-programming.html,
http://woboq.com/blog/internals-of-qmutex-in-qt5.html