

CS-453 (project)

Atomic primitives

Sébastien Rouault

Distributed Computing Laboratory

October 02, 2018

Last week

Summary

```

// Global var.
int a = 0;
int b = 0;

// Thread A
a = 1;
b = 1;

// Thread B
if (b == 1) {
    print(a, b);
    // a = 1, b = 1 
    // a = 1, b = 0 
    // a = 0, b = 1 
    // a = 0, b = 0 
}

```



Last week

Summary

```
// Global var.
```

```
#include <atomic>
```

```
using namespace std;
```

```
atomic<int> a = 0;
```

```
atomic<int> b = 0;
```

```
// Thread A
```

```
a.store(1, relaxed);
```

```
b.store(1, release);
```

```
// Thread B
```

```
auto v = b.load(acquire);
```

```
if (v == 1) {
```

```
    print(a.load(relaxed), v);
```

```
    // a = 1, b = 1 
```

```
    // a = 1, b = 0 
```

```
    // a = 0, b = 1 
```

```
    // a = 0, b = 0 
```

```
}
```

More atomic primitives

Overview

Name	C++ method(s)
Fetch-and-Add	fetch_add
Swap	exchange
Compare-and-Swap	compare_exchange_weak compare_exchange_strong

Limitation of fetch-and-add



Integral and **pointer** types only (C11, C++11)

Floating (and more) types added (since C++20)

More atomic primitives

Fetch-and-Add

```
#include <atomic>
using namespace std;
using Order = memory_order;

// Pseudo C++ code below
T atomic<T>::fetch_add(T v, Order order = seq_cst) {
    atomic {
        auto t = load(relaxed); // Fetch
        atomic_thread_fence(order);
        store(t + v, relaxed); // Add
        return t;
    }
}
```

More atomic primitives

Swap

```
#include <atomic>
using namespace std;
using Order = memory_order;

// Pseudo C++ code below
T atomic<T>::exchange(T v, Order order = seq_cst) {
    atomic {
        auto t = load(relaxed);
        atomic_thread_fence(order);
        store(v, relaxed); // Just overwrite
        return t;
    }
}
```

More atomic primitives

Compare-and-Swap

```
// [...]  
  
// Pseudo C++ code below  
bool atomic<T>::compare_exchange_strong(T& e, T v,  
                                         Order succ = seq_cst,  
                                         Order fail = success) {  
    atomic {  
        bool same = (load(relaxed) == e);  
        atomic_thread_fence(same ? succ : fail);  
        if (same)  
            store(v, relaxed);  
        else e = load(relaxed);  
        return same;  
    }  
}
```

More atomic primitives

Compare-and-Swap

```
// [...]  
  
// Pseudo C++ code below  
bool atomic<T>::compare_exchange_weak(T& e, T v,  
                                       Order succ = seq_cst,  
                                       Order fail = success) {  
    atomic {  
        bool same = (load(relaxed) == e);  
        // weak: 'same' may spuriously be false  
        atomic_thread_fence(same ? succ : fail);  
        if (same)  
            store(v, relaxed);  
        else e = load(relaxed);  
        return same;  
    }  
}
```


TP: my own (lightweight) mutex

Setup

1. Clone/download again or pull

```
https://github.com/LPD-EPFL/CS453-2018-project.git
```

2. Go to directory `playground`

3. Execute `$ make run` and you should see:

```
[...]
```

```
Hello from C++ version in thread .../...
```

```
Hello from C++ version in thread .../...
```

```
[...]
```

```
** Inconsistency detected (... != ...) **
```

- (4.) Comment `config.h:4` out and re-execute `$ make run`

Beyond

Thorough reference and more stellar blogs

- <https://en.cppreference.com/w/cpp/atomic/atomic>
- <https://preshing.com/20120226/roll-your-own-lightweight-mutex/>
- <https://cbloomrants.blogspot.com/2011/07/07-15-11-review-of-many-mutex.html>

Next time, project!

- Last presentation (*no slide, everything will be on the web*) about the transactional memory interface and your task
- FYI deadlines 23/11/18 23:59:59 (*step 1/2*)
20/12/18 23:59:59 (*step 2/2*)