CS-453 (project)

Atomic primitives

Sébastien Rouault

Distributed Computing Laboratory

October 01, 2019

Last week

Original code

```
// Global var.

int a = 0;
int b = 0;

// Thread A

a = 1; // write
b = 1; // write
```

```
// Thread B

auto v = b; // read
if (v == 1) {
    print(a, v); // read
    // a = 1, v = 1   ☑
    // a = 1, v = 0   ☐
    // a = 0, v = 1   ☑
    // a = 0, v = 0   ☐
}
```

Last week

Corrected code

```
// Global var.

#include <atomic>

int a = 0;
std::atomic<int> b = 0;

// Thread A

a = 1; // write
b.store(1, release);
```

```
// Thread B

auto v = b.load(acquire);
if (v == 1) {
    print(a, v); // read
    // a = 1, v = 1   ☑
    // a = 1, v = 0   ☐
    // a = 0, v = 1   ☐
    // a = 0, v = 0   ☐
}
```

More atomic primitives

Overview

Name	**C++ method(s)**
Read/write	`load`/`store`
Fetch–and–...	`fetch_...`
Swap	`exchange`
Compare–and–Swap	`compare_exchange_weak` `compare_exchange_strong`

Limitation of `fetch-and-...`

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

Floating (and more) types may be added (C++20)

More atomic primitives

Fetch–and–...

```
// Pseudo C++17 code below
#include <atomic>
using namespace std;
using Order = memory_order;

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

```
// Pseudo C++17 code below
#include <atomic>
using namespace std;
using Order = memory_order;

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++17 code below
bool atomic<T>::compare_exchange_strong(T& e, T v,
                    Order succ = seq_cst, Order fail = seq_cst) {
      atomic {
            bool same = (load(relaxed) == e);
            atomic_thread_fence(same ? succ : fail);
            if (same)
                  store(v, relaxed);
            else e = load(relaxed); // NB: e overwritten on failure
            return same;
      }
}
```

More atomic primitives

Compare-and-Swap

```
// [...]
// Pseudo C++17 code below
bool atomic<T>::compare_exchange_weak(T& e, T v,
                    Order succ = seq_cst, Order fail = seq_cst) {
     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); // NB: e overwritten on failure
          return same;
     }
}
```

TP: my own (lightweight) mutex

Setup

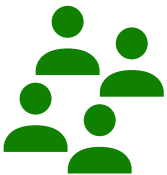
1. Checkout branch master from
   `https://github.com/LPD-EPFL/CS453-2019-project.git`
2. Go to directory `playground`
3. Execute `$ make run` and you should see:

   ```
   [...]
   Hello from thread .../...
   [...]
   ** Inconsistency detected (... != ...) **
   ```

4. Complete the 4 methods `Lock::...` in `entrypoint.cpp`, implementing your own lightweight mutex, then run again.

TP: my own (lightweight) mutex

The Analogy of the Talking Stick



4 threads

shared resource

mutex

- Speaking at the ***same time*** is forbidden
- Must ***acquire*** the talking stick to speak
- The other speakers ***wait for*** the talking stick

Resources — 1st link discusses (many) solutions...

- Charles Bloom — Review of many mutex implementations
- Jeff Preshing — Locks aren't slow, lock contention is
- Jeff Preshing — You can do any kind of atomic RMW ops.