std::exchange IDIOMS

BEN DEANE

bdeane@blizzard.com

SEPTEMBER 28TH, 2017

std::exchange

- Added in C++14
- Very simple
- If you are still on C++11, you can drop in an implementation

```
template<class T, class U = T>
constexpr T exchange(T& obj, U&& new_value) noexcept(noexcept(...))
{
    T old_value = std::move(obj);
    obj = std::forward<U>(new_value);
    return old_value;
}
```

(from cppreference.com - plus `noexcept` & `constexpr`)

ASYNC CODE

```
struct SomeAsyncSubSystem
{
  // this class gathers results, handles events
  // on a periodic tick it dispatches callbacks
  void tick();

  std::vector<Callback> m_callbacks;
};

void SomeAsyncSubSystem::tick()
{
  // iterate callbacks and dispatch them

  // but watch out for re-entrancy! clients tend to
  // register/unregister callbacks in response to being called...

  // so we'll use the swap-and-iterate idiom
}
```

PATTERN 1 (BEFORE)

```
void SomeAsyncSubSystem::tick()
{
  std::vector<Callback> v;
  std::swap(v, m_callbacks);
  for (const auto& callback : v)
  {
    callback();
  }
}
```

Single-threaded, simple re-entrancy protection.

PATTERN 1 (EVOLVED)

```
void SomeAsyncSubSystem::tick()
{
  for (const auto& callback : std::exchange(m_callbacks, {}))
  {
    callback();
  }
}
```

We saved a move and a variable.

PATTERN 2 (BEFORE)

```
void SomeAsyncSubSystem::tick()
{
  std::vector<Callback> v;
  std::swap(v, m_callbacks);
  PostToAnotherThread([v_ = std::move(v)] () {
    for (const auto& callback : v_)
    {
      callback();
    }
  }
}
```

Send stuff to another thread.

PATTERN 2 (EVOLVED)

```
void SomeAsyncSubSystem::tick()
{
  PostToAnotherThread([v_ = std::exchange(m_callbacks, {})] () {
    for (const auto& callback : v_)
    {
      callback();
    }
  }
}
```

Saved a move + variable again.

The result of `std::exchange` RVOs into the lambda capture.

PATTERN 3 (BEFORE)

Thread safety?

```
void SomeAsyncSubSystem::tick()
{
  std::vector<Callback> v;
  {
    std::lock_guard<std::mutex> lock(m_mutex);
    std::swap(v, m_callbacks);
  }

  for (const auto& callback : v)
  {
    callback();
  }
}
```

We like to scope locks as tightly as possible.

PATTERN 3 (EVOLVED)

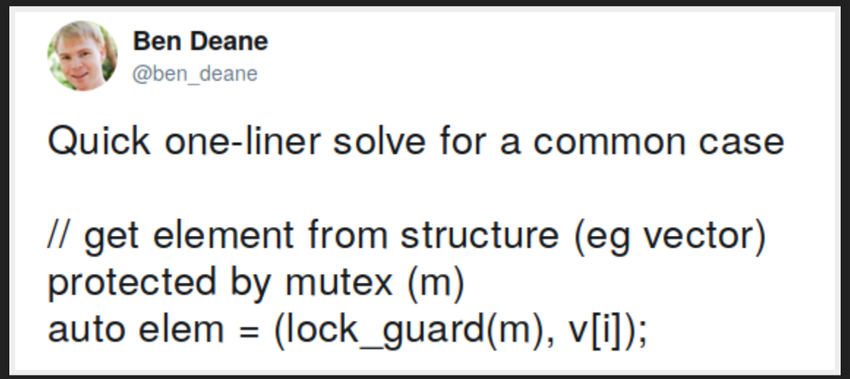
```
void SomeAsyncSubSystem::tick()
{
  auto v = (std::lock_guard<std::mutex>{m_mutex},
            std::exchange(m_callbacks, {}));

  for (const auto& callback : v)
  {
    callback();
  }
}
```

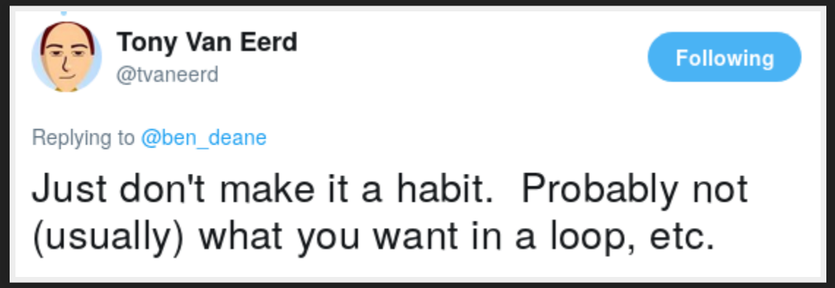
Comma operator FTW.

RULES OF C++

1. Don't pay for what you don't use.
2. `vector` is always the right choice.

WHERE `vector` IS NOT THE RIGHT CHOICE





THANKS

- start using `std::exchange` if you're not already
- more efficient
- no declaration/initialization split
- less code

The thread-safe swap-and-iterate idiom.

```
auto container = (lock_guard{mut}, exchange(other_container, {}));
for (auto& elem : container) {
  // ...
}
```