Some helper functions for interlocked pointer operations

devblogs.microsoft.com/oldnewthing/20151109-00

November 9, 2015

Raymond Chen

The pointer-related Interlocked functions all operate on `void*`. In practice, though, you are operating on typed pointers. Here are some helper functions to save you a bunch of typing.

```
template<typename T, typename U>
T* InterlockedExchangePointerT(
    T* volatile *target,
    U value)
{
  return reinterpret_cast<T*>(InterlockedExchangePointer(
    reinterpret_cast<void* volatile*>(target),
    static_cast<T*>(value)));
}

// Repeat for InterlockedExchangePointerAcquire and
// InterlockedExchangePointerNoFence.

template<typename T, typename U, typename V>
T* InterlockedCompareExchangePointerT(
    T* volatile *target,
    U exchange,
    V compare)
{
  return reinterpret_cast<T*>(InterlockedCompareExchangePointer(
    reinterpret_cast<void* volatile*>(target),
    static_cast<T*>(exchange),
    static_cast<T*>(compare)));
}

// Repeat for InterlockedCompareExchangePointerAcquire,
// InterlockedCompareExchangePointerRelease, and
// InterlockedCompareExchangePointerNoFence.
```

The naïve versions of these functions would be

```
template<typename T, typename U>
T* InterlockedExchangePointerT(
    T* volatile *target,
    T* value)
{
  return reinterpret_cast<T*>(InterlockedExchangePointer(
    reinterpret_cast<void* volatile*>(target),
    value));
}

template<typename T, typename U, typename V>
T* InterlockedCompareExchangePointerT(
    T* volatile *target,
    T* exchange,
    T* compare)
{
  return reinterpret_cast<T*>(InterlockedCompareExchangePointer(
    reinterpret_cast<void* volatile*>(target),
    exchange,
    compare));
}
```

but those simpler versions fail on things like

```
class Base { ... };
class Derived : public Base { ... };

extern Base* b;

Derived* d = new Derived();
if (InterlockedCompareExchange(&p, d, nullptr)) ...
```

because the compiler wouldn't be able to choose a value for `T`. From the first paramter, it would infer that `T = Base`; from the second parameter, it would infer that `T = Derived`; and from the third parameter, it would give up because it can't figure out what value of `T` would result in `T*` being the same as `std::nullptr_t`.

(You can guess how I discovered these limitations of the naïve versions.)

Raymond Chen

Follow