Subject: Re: Sharing and Locking
Posted by gridem on Mon, 08 Mar 2010 09:57:27 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 08 March 2010 03:42Well, you can say that, but it is a bit far-stretched IMO. Pte/Ptr are solely for solving dangling pointer issue. Unlike shared_ptr (correct me if I am wrong), Ptr can point to stack objects and most of time they really do.

intrusive_ptr can do the same thing, but it needs some additional steps to emulate the same behavior. From my point of view in MT application stack object may be destroyed at any time and Ptr/Pte can not prevent from using the already destroyed object:

```
struct Foo : Pte<Foo> {
    void SomeAction()  { INTERLOCKED { ... } }
};

Ptr<Foo> ptr;

// thread 1:
{
    Foo foo;
    ptr = &foo;
} // A1: foo have been destroyed

// thread 2
if (ptr) // A2
    ptr->SomeAction(); // A3
```

For example: thread 1 creates the object and ptr references to foo. Thread 2 checks that ptr has the reference and calls the method. We can suppose that SomeAction has internal Mutex to prevent simultanious access to class values. But if between A2 and A3 the foo have been destroyed (A1), than the race takes place and the application will be crashed.

May be I cannot understand how Pte/Ptr can be used correctly but shared_ptr can prevents from such situation in more atomical and strict manner.

luzr wrote on Mon, 08 March 2010 03:42I wonder how you can even do that?

The obvious way how to resolve the same problem is the following:

```
struct Ctrl
{
    struct Base; // implementation

    Ctrl() : base(new Base) {} // at cpp file
    bool IsForeground() const { return base->IsForeground(); } // at cpp file
    void SetForeground() { base->SetForeground(); } // at cpp file
    ...
```

```
protected:
    Ctrl(Base* b) : base(b) {} // at cpp file

private:
    shared_ptr<Base> base;
};

struct Pusher : Ctrl
{
    struct Base : Ctrl::Base { ... };

    Pusher() : Ctrl(new Base) {}
    ...
};
```

Ctrl has shared semantic and can be used as value in most cases (no need const references). This idiom guarantees that base will be available at any time and will be destroyed correctly.

luzr wrote on Mon, 08 March 2010 03:42Yes, this correct, Pte/Ptr is not great perfomance-wise. (OTOH, Mutex is just two atomic operations

In case when only one object acquire the lock it is true but if lock was acquired and someone wants to acquire the same lock than it takes much more time (thread sleeps until the lock will be released, so thread goes to kernel and from kernel, on Windows it's relatively heavy operation).
