
Subject: StaticMutex/ONCELOCK question
Posted by [Novo](#) on Tue, 03 Feb 2009 05:28:01 GMT
[View Forum Message](#) <> [Reply to Message](#)

I couldn't understand completely several things with StaticMutex and ONCELOCK.

StaticMutex will never call destructor of a contained Mutex object. Is this meant to be?

```
#define ONCELOCK \  
for(static volatile bool o_b_; !ReadWithBarrier(o_b_);) \  
for(static StaticMutex o_ss_; !o_b_;) \  
for(Mutex::Lock o_ss_lock__(o_ss_); !o_b_; BarrierWrite(o_b_, true))
```

How the above code actually works?

TIA

Subject: Re: StaticMutex/ONCELOCK question
Posted by [mirek](#) on Tue, 03 Feb 2009 06:41:36 GMT
[View Forum Message](#) <> [Reply to Message](#)

Novo wrote on Tue, 03 February 2009 00:28: I couldn't understand completely several things with StaticMutex and ONCELOCK.

StaticMutex will never call destructor of a contained Mutex object. Is this meant to be?

Yes. OS will clean that up when program exits.

Quote:

```
#define ONCELOCK \  
for(static volatile bool o_b_; !ReadWithBarrier(o_b_);) \  
for(static StaticMutex o_ss_; !o_b_;) \  
for(Mutex::Lock o_ss_lock__(o_ss_); !o_b_; BarrierWrite(o_b_, true))
```

How the above code actually works?

TIA

Do not get fooled by 3 'for' loops - these are just syntactic sugar to make ONCELOCK work on C statements and blocks - they in fact simulate the outer block

```
{
static volatile bool o_b_;
if(!ReadWithBarrier(o_b_)) {
    static StaticMutex mutex;
    mutex.Enter();
    {
        do_the_initialization - the statement 'body'
        BarrierWrite(o_b_);
    }
}
}
```

The purpose is to avoid locking mutex in subsequent passes of ONCELOCK - you need the barrier code to do that.

Note that both compilers we use optimize the for loops away.

Mirek

Subject: Re: StaticMutex/ONCELOCK question
Posted by [Novo](#) on Tue, 03 Feb 2009 19:28:05 GMT
[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Tue, 03 February 2009 01:41
Do not get fooled by 3 'for' loops - these are just syntactic sugar to make ONCELOCK work on C statements and blocks - they in fact simulate the outer block

Thanks. I understand the idea with loops. I'm using similar technique to handle transactions myself.

What I do not understand is how uninitialized o_b_ works.

```
{
static volatile bool o_b_;
if(!ReadWithBarrier(o_b_)) {
    ...
}
}
```

Subject: Re: StaticMutex/ONCELOCK question
Posted by [Novo](#) on Sun, 08 Feb 2009 02:09:19 GMT
[View Forum Message](#) <> [Reply to Message](#)

Novo wrote on Tue, 03 February 2009 14:28
What I do not understand is how uninitialized o_b_ works.

```
{  
    static volatile bool o_b_  
    if(!ReadWithBarrier(o_b_)) {  
        ...  
    }  
}
```

I finally figured out that myself. ANSI-compatible compiler initializes all static POD data with zeroes before a very first function call. So, this static variable is always initialized in a thread-safe way.

There is always something to learn about C.
