链滴

一个轻量级的单写多读锁

作者：localvar

原文链接：https://ld246.com/article/1521533355333

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

与《windows核心编程》上的那个相比最大的优势是体积小, 它只有四个字节(《windows核心编程》的那个至少是它的10倍), 如果你有大量对象需要进行单写多读访问的话, 它会比较适用. 缺点是它在加时使用的等待函数是Sleep, 如果访问冲突很多的话, 效率比较低. 代码如下, 很简单, 就不多做解释了:

```
//////////////////////////////////////////////////////////////////////////
// 头文件
#ifndef SWMR_LOCK_H
#define SWMR_LOCK_H

#ifndef SWMR_LOCK_NUMBER_OF_WRITER
#define SWMR_LOCK_NUMBER_OF_WRITER        1
#endif // SWMR_LOCK_NUMBER_OF_WRITER

typedef volatile long SWMR_LOCK

void SwmrLockInitialize( SWMR_LOCK* pLock );
void SwmrLockWriteLock( SWMR_LOCK* pLock );
void SwmrLockWriteUnlock( SWMR_LOCK* pLock );
void SwmrLockReadLock( SWMR_LOCK* pLock );
void SwmrLockReadUnlock( SWMR_LOCK* pLock );
void SwmrLockUninitialize( SWMR_LOCK* pLock );

#endif

//////////////////////////////////////////////////////////////////////////
// 实现文件
#include "srmrl.h"

//////////////////////////////////////////////////////////////////////////

#define WRITING_FLAG    0x40000000

//////////////////////////////////////////////////////////////////////////

void SwmrLockInitialize( SWMR_LOCK* pLock )
{
   *pLock = 0;
}

//////////////////////////////////////////////////////////////////////////

void SwmrLockWriteLock( SWMR_LOCK* pLock )
{
   long old, xchg;
   do {
      old = *pLock;
#if( SWMR_LOCK_NUMBER_OF_WRITER > 1 )
      if( old & WRITING_FLAG )
      {
         Sleep( 0 );
         continue;
      }
#endif // ( SWMR_LOCK_NUMBER_OF_WRITER > 1 )
      xchg = old | WRITING_FLAG;
```

```
    } while( _InterlockedCompareExchange( pLock, xchg, old ) != old );

    // wait until all readers quit reading
    while( old != WRITING_FLAG )
    {
        Sleep( 0 );
        old = *pLock;
    }
}

//////////////////////////////////////////////////////////////////////////////

void SwmrLockWriteUnlock( SWMR_LOCK* pLock )
{
    *pLock = 0;
}

//////////////////////////////////////////////////////////////////////////////

void SwmrLockReadLock( SWMR_LOCK* pLock )
{
    long old, xchg;
    do {
        old = *pLock;
        if( old & WRITING_FLAG )
        {
            Sleep( 0 );
            continue;
        }
        xchg = old + 1;
    } while( _InterlockedCompareExchange( pLock, xchg, old ) != old );
}

//////////////////////////////////////////////////////////////////////////////

void SwmrLockReadUnlock( SWMR_LOCK* pLock )
{
    _InterlockedDecrement( pLock );
}

//////////////////////////////////////////////////////////////////////////////

void SwmrLockUninitialize( SWMR_LOCK* pLock )
{
    pLock; // has nothing to do
}

//////////////////////////////////////////////////////////////////////////////
```