平进水坚圆蛋的专栏

其实不想走

导航

博客园
首　页
新随笔
联　系
订　阅 XML
管　理

<		2010年9月				>
日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

公告

博客很久没更新了,以前写的内容均是实践性内容.如果对我的博客有疑问或是想咨询的可以关注weibo.com/1021506631 @teof 博主做过的内容偏杂不深.做过VC++,.Net/C#,互联网服务器端,php,运维(mediawiki,redmin,nagios),新浪微博平台接口接入和开发,游戏服务器端以及手游平台各种接入,unity3d和ngui,个人最熟悉的是Erlang.热爱计算可惜技术的各个方面都不熟悉. 本科毕

详解volatile在C++中的作用

volatile的介绍

volatile类似于大家所熟知的const也是一个类型修饰符。volatile是给编译器的指示来说明对它所修饰的对象不应该执行优化。volatile的作用就是用来进行多线程编程。在单线程中那就是只能起到限制编译器优化的作用。所以单线程的童鞋们就不用浪费精力看下面的了。

没有volatile的结果

如果没有volatile，你将无法在多线程中并行使用到基本变量。下面举一个我开发项目的实例（这个实例采用的是C#语言但不妨碍我们讨论C++）。在学校的一个.Net项目的开发中，我曾经在多线程监控中用到过一个基本变量Int32型的，我用它来控制多线程中监控的一个条件。考虑到基本变量是编译器自带的而且无法用lock锁上，我想当然的以为是原子操作不会有多线程的问题，可实际运行后发现程序的运行有时正常有时异常，改为用Dictionary对象处理并加锁以后才彻底正常。现在想来应该是多线程同时操作该变量了，具体的将在下面说清。

volatile的作用

如果一个基本变量被volatile修饰，编译器将不会把它保存到寄存器中，而是每一次都去访问内存中实际保存该变量的位置上。这一点就避免了没有volatile修饰的变量在多线程的读写中所产生的由于编译器优化所导致的灾难性问题。所以多线程中必须要共享的基本变量一定要加上volatile修饰符。当然了，volatile

业于合肥工业大学信息与计算科学.研究生毕业于北京航空航天大学计算机学院软件工程研究所.曾工作于北京淘宝量子统计后端,红岩凯鹏信息科技有限公司帮老板创业过(服务器负责人),现在广州爱游信息科技有限公司(4399)山水界担任Erlang研发工程师.

昵称：teof
园龄：4年11个月
粉丝：4
关注：1
+加关注

还能让你在编译时期捕捉到非线程安全的代码。我在下面还会介绍一位大牛使用智能指针来顺序化共享区代码的方法，在此对其表示感谢。

泛型编程中曾经说过编写异常安全的代码是很困难的，可是相比起多线程编程的困难来说这就太小儿科了。多线程编程中你需要证明它正确，需要去反复地枯燥地调试并修复，当然了，资源竞争也是必须注意的，最可恨的是，有时候编译器也会给你点颜色看看。。。

```
class Student
{
public:
    void Wait() //在北航排队等吃饭实在是很痛苦的事情。。。
    {
        while (!flag)
        {
            Sleep(1000); // sleeps for 1000 milliseconds
        }
    }
    void eat()
    {
        flag = true;
    }
    ...
private:
    bool flag;
};
```

好吧，多线程中你就等着吃饭吧，可在这个地方估计你是永远等不到了，因为flag被编译器放到寄存器中去了，哪怕在你前面的那位童鞋告诉你flag=true了，可你就好像瞎了眼看不到这些了。这么诡异的情况

你的成功在于你每天养成的习惯
人
谈谈个人对SnmpSharpNet的实践经验
有道笔试
云计算电话面试

随笔档案

2013年6月 (1)
2012年9月 (1)
2012年5月 (2)
2010年9月 (3)

阅读排行榜

1. 详解volatile在C++中的作用(3853)
2. 谈谈个人对SnmpSharpNet的实践经验(1248)
3. 第一次开发软件个人总结(1043)
4. 关于C#定时器的总结(79)
5. Erlang——hotwheels源码剖析(47)

评论排行榜

1. 谈谈个人对SnmpSharpNet的实践经验(4)

的发生时因为你所用到的判断值是之前保存到寄存器中的，这样原来的地址上的flag值更改了你也没有获取。该怎么办呢？对了，改成volatile就解决了。

volatile对基本类型和对用户自定义类型的使用与const有区别，比如你可以把基本类型的non-volatile赋值给volatile，但不能把用户自定义类型的non-volatile赋值给volatile，而const都是可以的。还有一个区别就是编译器自动合成的复制控制不适用于volatile对象，因为合成的复制控制成员接收const形参，而这些形参又是对类类型的const引用，但是不能将volatile对象传递给普通引用或const引用。

如何在多线程中使用好volatile

在多线程中，我们可以利用锁的机制来保护好资源临界区。在临界区的外面操作共享变量则需要volatile，在临界区的里面则non-volatile了。我们需要一个工具类LockingPtr来保存mutex的采集和volatile的利用const_cast的转换（通过const_cast来进行volatile的转换）。

首先我们声明一个LockingPtr中要用到的Mutex类的框架：

```
class Mutex
{
public:
    void Acquire();
    void Release();
    ...
};
```

接着声明最重要的LockingPtr模板类：

```
template <typename T>
class LockingPtr {
public:
```

```
    // Constructors/destructors
    LockingPtr(volatile T& obj, Mutex& mtx)
        : pObj_(const_cast<T*>(&obj)),
         pMtx_(&mtx)
    {    mtx.Lock();    }
    ~LockingPtr()
    {    pMtx_->Unlock();    }
    // Pointer behavior
    T& operator*()
    {    return *pObj_;    }
    T* operator->()
    {   return pObj_;   }
private:
    T* pObj_;
    Mutex* pMtx_;
    LockingPtr(const LockingPtr&);
    LockingPtr& operator=(const LockingPtr&);
};
```

尽管这个类看起来简单，但是它在编写争取的多线程程序中非常的有用。你可以通过对它的使用来使得对多线程中共享的对象的操作就好像对volatile修饰的基本变量一样简单而且从不会使用到const_cast。下面来给一个例子：

假设有两个线程共享一个vector<char>对象：

```
class SyncBuf {
public:
    void Thread1();
```

```
    void Thread2();
private:
    typedef vector<char> BufT;
    volatile BufT buffer_;
    Mutex mtx_; // controls access to buffer_
};
```

在函数Thread1中，你通过lockingPtr<BufT>来控制访问buffer_成员变量：

```
void SyncBuf::Thread1() {
    LockingPtr<BufT> lpBuf(buffer_, mtx_);
    BufT::iterator i = lpBuf->begin();
    for (; i != lpBuf->end(); ++i) {
        ... use *i ...
    }
}
```

这个代码很容易编写和理解。只要你需要用到buffer_你必须创建一个lockingPtr<BufT>指针来指向它，并且一旦你这么做了，你就获得了容器vector的整个接口。而且你一旦犯错，编译器就会指出来：

```
void SyncBuf::Thread2() {
    // Error! Cannot access 'begin' for a volatile object
    BufT::iterator i = buffer_.begin();
    // Error! Cannot access 'end' for a volatile object
    for (; i != lpBuf->end(); ++i) {
        ... use *i ...
    }
}
```

这样的话你就只有通过const_cast或LockingPtr来访问成员函数和变量了。这两个方法的不同之处在于后者提供了顺序的方法来实现而前者是通过转换为volatile来实现。LockingPtr是相当好理解的，如果你需要调用一个函数，你就创建一个未命名的暂时的LockingPtr对象并直接使用：

```
unsigned int SyncBuf::Size() {
    return LockingPtr<BufT>(buffer_, mtx_)->size();
}
```

LockingPtr在基本类型中的使用

在上面我们分别介绍了使用volatile来保护对象的意外访问和使用LockingPtr来提供简单高效的多线程代码。现在来讨论比较常见的多线程处理共享基本类型的一种情况：

```
class Counter
{
public:
    ...
    void Increment() { ++ctr_; }
    void Decrement() { —-ctr_; }
private:
    int ctr_;
};
```

这个时候可能大家都能看出来问题所在了。1.ctr_需要是volatile型。2.即便是++ctr_或--ctr_，这在处理中仍是需要三个原子操作的（Read-Modify-Write）。基于上述两点，这个类在多线程中会有问题。现在我们就来利用LockingPtr来解决：

```
class Counter
{
```

```
public:
    ...
    void Increment() { ++*LockingPtr<int>(ctr_, mtx_); }
    void Decrement() { –?*LockingPtr<int>(ctr_, mtx_); }
private:
    volatile int ctr_;
    Mutex mtx_;
};
```

volatile成员函数

关于类的话，首先如果类是volatile则里面的成员都是volatile的。其次要将成员函数声明为volatile则同const一样在函数最后声明即可。当你设计一个类的时候，你声明的那些volatile成员函数是线程安全的，所以那些随时可能被调用的函数应该声明为volatile。考虑到volatile等于线程安全代码和非临界区；non-volatile等于单线程场景和在临界区之中。我们可以利用这个做一个函数的volatile的重载来在线程安全和速度优先中做一个取舍。具体的实现此处就略去了。

总结

在编写多线程程序中使用volatile的关键四点：

1.将所有的共享对象声明为volatile；

2.不要将volatile直接作用于基本类型；

3.当定义了共享类的时候，用volatile成员函数来保证线程安全；

4.多多理解和使用volatile和LockingPtr！（强烈建议）

« 上一篇：关于C#定时器的总结
» 下一篇：第一次开发软件个人总结

posted on 2010-09-19 19:03 teof 阅读(3853) 评论(0) 编辑 收藏

最新IT新闻:

· 百度为移动转型付出代价
· 为何我不在哈佛大学当教授而要去 Google 工作
· 梅姐：Tumblr增速超Instagram成发展最快的社交网络
· 思科3成经理层人员被换掉了
· 这就是英国第一辆无人驾驶汽车
» 更多新闻...

最新知识库文章:

· 使用2-3法则设计分布式数据访问层
· 数据清洗经验
· 设计中的变与不变
· 通俗解释「为什么数据库难以拓展」
· 手机淘宝高质量持续交付探索之路
» 更多知识库文章...