Chinese translated version of Documentation/volatile-considered-harmful.txt

If you have any comment or update to the content, please contact the
original document maintainer directly. However, if you have a problem
communicating in English you can also ask the Chinese maintainer for
help. Contact the Chinese maintainer if this translation is outdated
or if there is a problem with the translation.

Maintainer: Jonathan Corbet <corbet@lwn.net>
Chinese maintainer: Bryan Wu <bryan.wu@analog.com>

Documentation/volatile-considered-harmful.txt 的中文翻译

如果想评论或更新本文的内容，请直接联系原文档的维护者。如果你使用英文
交流有困难的话，也可以向中文版维护者求助。如果本翻译更新不及时或者翻
译存在问题，请联系中文版维护者。

英文版维护者： Jonathan Corbet <corbet@lwn.net>
中文版维护者： 伍鹏 Bryan Wu <bryan.wu@analog.com>
中文版翻译者： 伍鹏 Bryan Wu <bryan.wu@analog.com>
中文版校译者： 张汉辉 Eugene Teo <eugeneteo@kernel.sg>
                杨瑞 Dave Young <hidave.darkstar@gmail.com>
以下为正文

为什么不应该使用“volatile”类型

C程序员通常认为volatile表示某个变量可以在当前执行的线程之外被改变；因此，在内核中用到共享数据结构时，常常会有C程序员喜欢使用volatile这类变量。换句话说，他们经常会把volatile类型看成某种简易的原子变量，当然它们不是。在内核中使用volatile几乎总是错误的；本文档将解释为什么这样。

理解volatile的关键是知道它的目的是用来消除优化，实际上很少有人真正需要这样的应用。在内核中，程序员必须防止意外的并发访问破坏共享的数据结构，这其实是一个完全不同的任务。用来防止意外并发访问的保护措施，可以更加高效的避免大多数优化相关的问题。

像volatile一样，内核提供了很多原语来保证并发访问时的数据安全（自旋锁，互斥量,内存屏障等等），同样可以防止意外的优化。如果可以正确使用这些内核原语，那么就没有必要再使用volatile。如果仍然必须使用volatile，那么几乎可以肯定在代码的某处有一个bug。在正确设计的内核代码中，volatile能带来的仅仅是使事情变慢。

思考一下这段典型的内核代码：

```
    spin_lock(&the_lock);
    do_something_on(&shared_data);
    do_something_else_with(&shared_data);
    spin_unlock(&the_lock);
```

如果所有的代码都遵循加锁规则，当持有the_lock的时候，不可能意外的改变shared_data的
值。任何可能访问该数据的其他代码都会在这个锁上等待。自旋锁原语跟内存屏障一样——它
们显式的用来书写成这样 —— 意味着数据访问不会跨越它们而被优化。所以本来编译器认为

它知道在shared_data里面将有什么，但是因为spin_lock()调用跟内存屏障一样，会强制编译器忘记它所知道的一切。那么在访问这些数据时不会有优化的问题。

如果shared_data被声名为volatile，锁操作将仍然是必须的。就算我们知道没有其他人正在
使用它，编译器也将被阻止优化对临界区内shared_data的访问。在锁有效的同时，
shared_data不是volatile的。在处理共享数据的时候，适当的锁操作可以不再需要
volatile —— 并且是有潜在危害的。

volatile的存储类型最初是为那些内存映射的I/O寄存器而定义。在内核里，寄存器访问也应
该被锁保护，但是人们也不希望编译器“优化”临界区内的寄存器访问。内核里I/O的内存访问
是通过访问函数完成的；不赞成通过指针对I/O内存的直接访问，并且不是在所有体系架构上
都能工作。那些访问函数正是为了防止意外优化而写的，因此，再说一次，volatile类型不是必需的。

另一种引起用户可能使用volatile的情况是当处理器正忙着等待一个变量的值。正确执行一个忙等待的方法是：

```
    while (my_variable != what_i_want)
        cpu_relax();
```

cpu_relax()调用会降低CPU的能量消耗或者让位于超线程双处理器；它也作为内存屏障一样出
现，所以，再一次，volatile不是必需的。当然，忙等待一开始就是一种反常规的做法。

在内核中，一些稀少的情况下volatile仍然是有意义的：

- 在一些体系架构的系统上，允许直接的I/O内存访问，那么前面提到的访问函数可以使用
  volatile。基本上，每一个访问函数调用它自己都是一个小的临界区域并且保证了按照
  程序员期望的那样发生访问操作。

- 某些会改变内存的内联汇编代码虽然没有什么其他明显的附作用，但是有被GCC删除的可
  能性。在汇编声明中加上volatile关键字可以防止这种删除操作。

- Jiffies变量是一种特殊情况，虽然每次引用它的时候都可以有不同的值，但读jiffies
  变量时不需要任何特殊的加锁保护。所以jiffies变量可以使用volatile，但是不赞成
  其他跟jiffies相同类型变量使用volatile。Jiffies被认为是一种“愚蠢的遗留物”
  （Linus的话）因为解决这个问题比保持现状要麻烦的多。

- 由于某些I/O设备可能会修改连续一致的内存,所以有时,指向连续一致内存的数据结构
  的指针需要正确的使用volatile。网络适配器使用的环状缓存区正是这类情形的一个例
  子，其中适配器用改变指针来表示哪些描述符已经处理过了。

对于大多代码，上述几种可以使用volatile的情况都不适用。所以，使用volatile是一种
bug并且需要对这样的代码额外仔细检查。那些试图使用volatile的开发人员需要退一步想想
他们真正想实现的是什么。

非常欢迎删除volatile变量的补丁 — 只要证明这些补丁完整的考虑了并发问题。

注释

---

[1] http://lwn.net/Articles/233481/
[2] http://lwn.net/Articles/233482/

致谢

最初由Randy Dunlap推动并作初步研究
由Jonathan Corbet撰写
参考Satyam Sharma，Johannes Stezenbach，Jesper Juhl，Heikki Orsila，
H. Peter Anvin，Philipp Hahn和Stefan Richter的意见改善了本档。