**内存屏障详解**

当你看到“内存屏障”四个字的时候，你的第一反应是什么？寄存器里取出了错误的值？ifence,sfence之类的指令？还是诸如volatile之类的关键字？好吧，我第一次看到这四个字的时候，脑子里浮现出的是魔兽争霸里绿油油的铺满苔藓的岩石屏障－ －#，并且，当我搞明白内存屏障具体是什么，而且自认为对其很熟悉之后，我的第一反应依然是那几块绿油油的石头，而且很想上去A一把！

言归正传，先解释下什么是内存屏障。内存屏障是指“由于编译器的优化和缓存的使用，导致对内存的写入操作不能及时的反应出来，也就是说当完成对内存的写入操作之后，读取出来的可能是旧的内容”（摘自[《独辟蹊径品内核》](http://www.amazon.cn/exec/obidos/ASIN/B002JM12S6/spongeliu-23" \t "_blank)）。（这里概念貌似不是很准确，正确的定义：为了防止编译器和硬件的不正确优化，使得对存储器的访问顺序（其实就是变量）和书写程序时的访问顺序不一致而提出的一种解决办法。 它不是一种错误的现象，而是一种对错误现象提出的解决方发－－－－欢迎指正！！）

概念就是概念，生硬的东西，懂的人能从中悟出点什么，不懂的人还是一头雾水。不要着急，我们先给内存屏障分下类，然后挨个来研究一番，等看完这篇文章，再回来读读概念，你就懂了！

**内存屏障的分类：**

1. 编译器引起的内存屏障
2. 缓存引起的内存屏障
3. 乱序执行引起的内存屏障

**1、编译器引起的内存屏障：**

我们都知道，从寄存器里面取一个数要比从内存中取快的多，所以有时候编译器为了编译出优化度更高的程序，就会把一些常用变量放到寄存器中，下次使用该变量的时候就直接从寄存器中取，而不再访问内存，这就出现了问题，当其他线程把内存中的值改变了怎么办？也许你会想，编译器怎么会那么笨，犯这种低级错误呢！是的，编译器没你想象的那么聪明！让我们看下面的代码：（代码摘自[《独辟蹊径品内核》](http://www.amazon.cn/exec/obidos/ASIN/B002JM12S6/spongeliu-23)）

int flag=0;

void wait(){

while ( flag == 0 )

sleep(1000);

......

}

void wakeup(){

flag=1;

}

这段代码表示一个线程在循环等待另一个线程修改flag。 Gcc等编译器在编译的时候发现，sleep()不会修改flag的值，所以，为了提高效率，它就会把某个寄存器分配给flag，于是编译后就生成了这样的伪汇编代码：

void wait(){

movl flag, %eax;

while ( %eax == 0)

sleep(1000);

}

这时，当wakeup函数修改了flag的值，wait函数还在傻乎乎的读寄存器的值而不知道其实flag已经改变了，线程就会死循环下去。由此可见，编译器的优化带来了相反的效果！

但是，你又不能说是让编译器放弃这种优化，因为在很多场合下，这种优化带来的性能是十分可观的！那我们该怎么办呢？有没有什么办法可以避免这种情况？答案必须是肯定的，我们可以使用关键字volatile来避免这种情况。

volatile int flag = 0;

这样，我们就能避免编译器把某个寄存器分配给flag了。

好，上面所描述这些，就叫做“编译器优化引起的内存屏障”，是不是懂了点什么？再回去看看概念？

**2、缓存引起的内存屏障**

好，既然寄存器能够引起这样的问题，那么缓存呢？我们都知道，CPU会把数据取到一个叫做cache的地方，然后下次取的时候直接访问cache，写入的时候，也先将值写入cache。

那么，先让我们考虑，在单核的情况下会不会出现问题呢？先想一下，单核情况下，除了CPU还会有什么会修改内存？对了，是外部设备的DMA！那么，DMA修改内存，会不会引起内存屏障的问题呢？答案是，在现在的体系结构中，不会。

当外部设备的DMA操作结束的时候，会有一种机制保证CPU知道他对应的缓存行已经失效了；而当CPU发动DMA操作时，在想外部设备发送启动命令前，需要把对应cache中的内容写回内存。在大多数RISC的架构中，这种机制是通过一写个特殊指令来实现的。在X86上，采用一种叫做总线监测技术的方法来实现。就是CPU和外部设备访问内存的时候都需要经过总线的仲裁，有一个专门的硬件模块用于记录cache中的内存区域，当外部设备对内存写入的时候，就通过这个硬件来判断下改内存区域是否在cache中，然后再进行相应的操作。

那么，什么时候才能产生cache引起的内存屏障呢？多CPU？ 是的，在多CPU的系统里面，每个CPU都有自己的cache，当同一个内存区域同时存在于两个CPU的cache中时，CPU1改变了自己cache中的值，但是CPU2却仍然在自己的cache中读取那个旧值，这种结果是不是很杯具呢？因为没有访存操作，总线也是没有办法监测的，这时候怎么办？

对阿，怎么办呢？我们需要在CPU2读取操作之前使自己的cache失效，x86下，很多指令能做到这点，如lock前缀的指令，cpuid, iret等。内核中使用了一些函数来完成这个功能：mb(), rmb(), wmb()。用的也是以上那些指令，感兴趣可以去看下内核代码。

**3、乱序执行引起的内存屏障：**

我们都知道，超标量处理器越来越流行，连龙芯都是四发射的。超标量实际上就是一个CPU拥有多条独立的流水线，一次可以发射多条指令，因此，很多允许指令的乱序执行，具体怎么个乱序方法，可以去看体系结构方面的书，这里只说内存屏障。

指令乱序执行了，就会出现问题，假设指令1给某个内存赋值，指令2从该内存取值用来运算。如果他们两个颠倒了，指令2先从内存中取值运算，是不是就错了？

对于这种情况，x86上专门提供了lfence，sfence,和mfence 指令来停止流水线：

lfence:停止相关流水线，知道lfence之前对内存进行的读取操作指令全部完成

sfence:停止相关流水线，知道lfence之前对内存进行的写入操作指令全部完成

mfence:停止相关流水线，知道lfence之前对内存进行的读写操作指令全部完成

好，将完这三种类型，再回去看看概念，清晰了么？如果还不明白，那就是我的表达能力太有限了，自己网上再搜搜把！