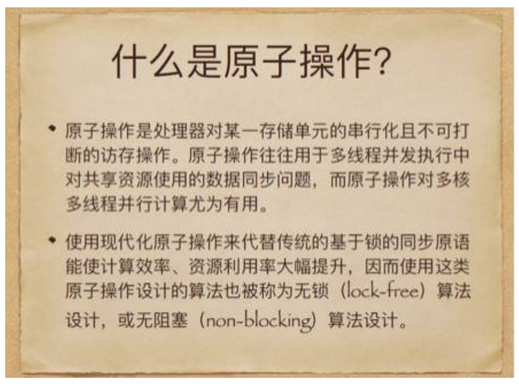
关于C11标准中原子操作，看这篇就够了！

2016-10-31 19:01 | CocoaChina

6月18日，由CocoaChina主办，Intel独家赞助的以“那些开发领域的新玩意”为主题的CVP系列开发者沙龙圆满落幕。沙龙上，CocoaChina社区名版主zenny_chen带来了《C11标准中原子操作简介》为主题的精彩分享。

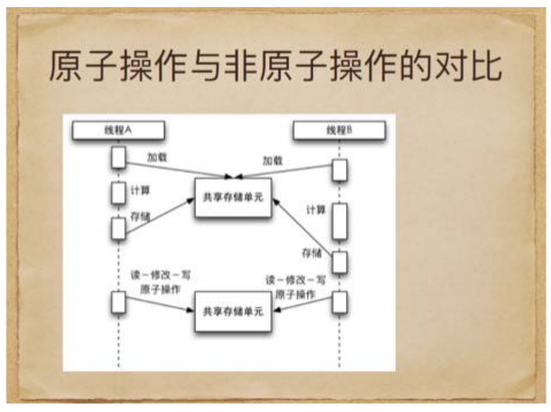
以下为演讲实录：

zenny_chen：很高兴跟大家分享C11标准，有很多的朋友可能不知道什么是C11标准，C11标准其实就是iSO/IEC C语言国际标准委员会在2011年发布的C语言官方标准，将其命名为ISO/IEC 9899:2011，我们将它简称为C11。不过我今天讲的这个C11标准原子操作，其实在C++编程语言中也是完全适用的，它们的API是完全一样的，所以C11标准中的原子使用范围比较多，此外像OpenCL 2.0也支持，以及Apple最新的Metal API也支持C11原子操作。



我这次为什么要选择C11标准原子操作呢？我们知道大数据、机器学**，以及现在VR、AR技术都非常火热，今年又被各大媒体称为VR元年，这些领域包括VR、人工智能、机器学**等其实都是属于高性能计算领域。在高性能并行计算领域中，我们往往会面对拥有多个核心的处理器的多线程并行计算，甚至利用像Intel Xeon phi众核加速器或GPU来做数据密集处理。在多核心多线程并行计算当中，有那么多线程将同时对数据进行处理，其中就会碰到一个非常重要的问题，就是数据一致性问题！当多个线程对同一个数据对象进行操作的时候，我们要保证对此数据修改结果的一致性，所以如何既能准确无误、又能够把系统负荷降到最低以保证数据一致性，那么这就要靠我们现代化的原子操作去达到这个目的。

那么传统的数据同步的方法实际上都是基于锁，比如互斥体、信号量、临界区等等。而原子操作形式跟基于锁的同步方法会有一些不太一样。我们现在来看一个对比，假设线程A和B全都是执行在一个多核处理器上（比如双核处理器上），并且两个线程同时进行，顺序图上我们可以看到线程A先对一个共享存储单元进行操作，先加载存储单元里面的值，然后对它计算，之后再进行存储；而与此同时，线程B也是在线程A加载后稍晚那么一丁点，也在对这个共享存储进行加载操作，做它的计算，最后存储。这两个线程做下来之后，比如说我们没有用原子操作，加载、计算和存储这三个操作之间是有缝隙的，所以这会导致像线程A，比如假定线程A已完成加载计算存储三个操作并将这个值写入，线程B也是完成了这个过程，那么共享存储单元最后的计算结果就是线程B的计算结果，不是线程B基于线程A操作之后的结果，这就会导致数据不完整性。



举个直观的例子：我现在共享存储单元里面的数字是1，线程A先把这个D加载进来，我做++操作就变成2，最后我把这个C11标准写到共享存储单元里；与此同时，线程B在另外一个核上同时执行，加载进来也是1，计算++操作不是+1，而是+10，这个值也存储进去，由于线程A先完成存储，然后是线程B完成存储，最终两个结束以后，共享存储单元值不是12，而是11，所以这就导致数据不一致性。如果线程B这个是快，线程A慢，最终的结果是2，也不是12。原子操作就不一样了。如果是采用现代化原子操作，加载、计算和存储就作为一整个不可分割且不可被打断的一个操作，此时如果这边是一个原子加法操作，线程A是+1操作，线程B是原子+10操作，那么当两个线程全都完成执行后，结果肯定就是12；如果线程A快，线程B慢，线程A在对共享存储单元做原子访问请求的时候，线程B如果同时要对它修改，那么会临时阻塞住，直到线程A对共享存储单元的操作完成以后，存储器控制器才会允许线程B的原子操作的修改请求，这样我们能够保证共享存储单元操作的数据和执行。

这边已经讲了几个数据同步问题的传统方法，比如说像互斥体，就是一个资源同时只能有一个线程进行访问，这个时候我们会用互斥体。还有一个信号量semaphore，大家学过操作系统应该比较熟，普遍用于生产者消费者问题，我可能会有多个资源多个线程都可以共享，生产者将信号量加1，消费者减1，比如减到小于0当前线程就被阻塞，当生产者再生产出一个资源，向那些阻塞的线程发出信号，让他们重新激活。还有一个就是临界区，就是类似于Objective-C中的一个@synchronized { }语句块。传统同步的特点就是对于单核单处理，多线程，尤其是单片机上面比较简单的处理器，一般处理器不需要提供原子操作，因为只要通过简单的开关中断操作即可对锁对象做原子性的修改操作。单核单处理器同时只能有一个线程进行执行，它的执行引擎是单个的，所以不会存在两个线程同时执行的情况。

对于多核多处理器环境下的数据同步，如果我要用开关中断来实现原子性是不切合实际的。现在做应用的同学应该比较熟悉，比如说当前线程就管当前线程做的事情，当前线程要对其他线程进行干扰，比如防止对它进行调度，这个显然是开销非常大的，而且也是不切实际，那么这个时候就只能通过原子操作进行多核多线程的数据同步。

我们再看一下上面的这张图。如果我们要线程A和线程B在单核单处理器的情况下，那么只要加载的时候加一个锁，如果线程A先执行，这个锁在它访问时是开的，在要用这个资源之前就把这个锁关上，然后做加载、计算和存储，即便我当前在做加载完成了或者计算完成的时候，线程A被操作系统调度出去了，把线程B调度进来，B这边也会面对这把锁，当它要访问下面资源的时候就会被锁住，同时当前线程会被挂起，等到A执行的时候，最后存储结果完了以后，把锁打开，线程B就可以从这里继续执行下去。而多个处理器，就像我们刚才讲的，我这边线程A对线程B是没法进行任何干扰的，这两个线程是完全同时执行的，所以这个时候你要用传统的锁是无法同步的，在多核处理器环境下，锁的实现也需要基于原子操作。

下面我们就介绍一下C11标准中的原子对象类型。请看Keynote：

最新文章

人的朴素，源于自信

养胃的最佳答案在这里。

窗帘的5种挂法，你知道吗？

香槟酒是属于什么类型葡萄酒

【智慧生活】长寿者的8个健康理念！

女人外阴长了一个小疙瘩

题材书法提升中国书法艺术的五大亮点

第7讲 孙过庭职业辩析

唐卡居然是用这些画出来的，简直美翻了！

【珍贵视频】第十六世噶玛巴珍贵视频和图...

相关推荐

女人裸睡的好处， 常裸睡真能防病又美肤

【岁月凝华，共鉴珍品】 ——宝能太古城...

九里峰山四期花漫里湖山洋房诚意登记中

江山里26#楼二批新品即将加推 认筹存5千...

汉泰上上城8#楼109-206㎡新品火爆加推...

女子从11楼跳下，竟看到10楼恩爱夫妻正...

嘉福金融中心183㎡瞰江美宅 品味格调人生

海亮天城26#楼96-125㎡南向央景三四房 ...

同一款车，为何你的油耗是别人2倍？99%...

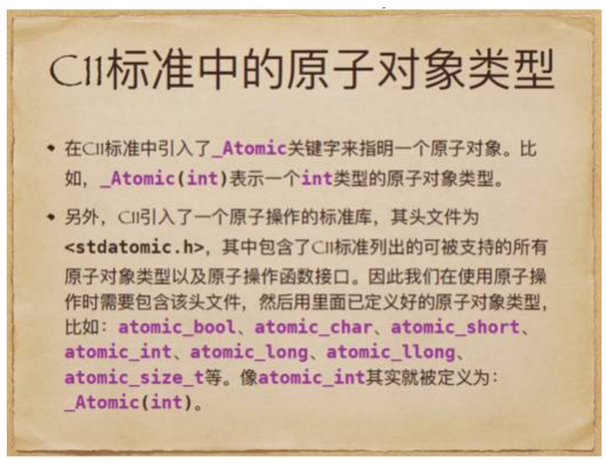
【第5大道】约58-62㎡轻五星级产权式酒...

对这篇文章不满意？

您可以继续搜索：

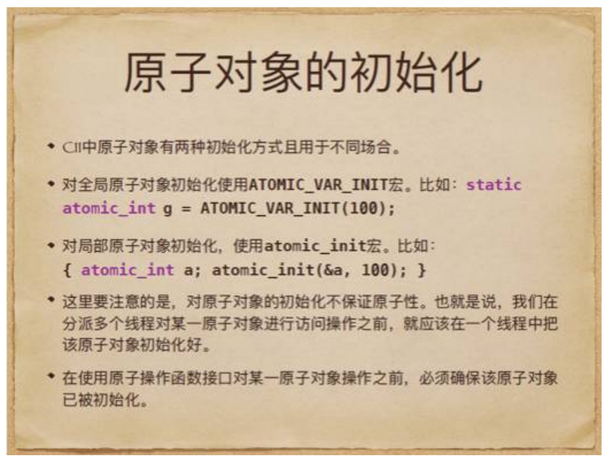
百度：关于C11标准中原子操作，看这篇就...

搜狗：关于C11标准中原子操作，看这篇就...



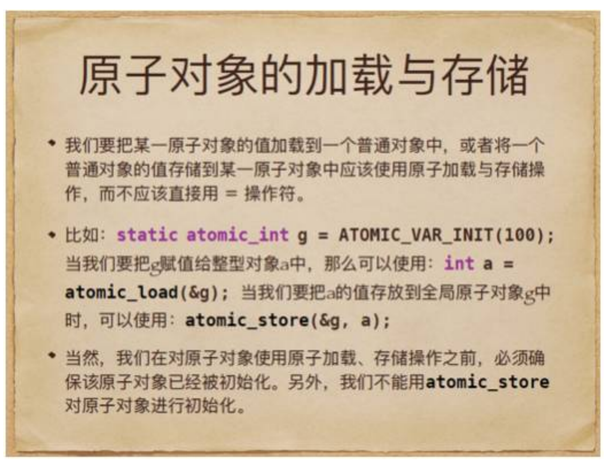
各位在使用原子操作的时候，尤其在GPU上面一定要当心，一般像GPU或者规范里面都会有写，它只能支持哪些数据类型，比如Metal里面只支持int，不支持其他的。现在基本上原子对象都是大家看到的，全都是属于整数类型范畴里面的。另外上述的int对应的无符号都是支持的，比如，无符号原子整型就是atomic-uint等等。

原子对象的初始化有两种方式，并且这两种方式是应用于两种不同场合的。请看Keynote:



这里，大家要注意原始对象初始化过程本身是不保证对原子对象的原子性操作，我们要用原子对象做多核多线程的一个同步之前，就是说先要把它初始化，然后再分派多个线程对它进行操作。

下面谈一下原子对象的加载与存储，请看Keynote：

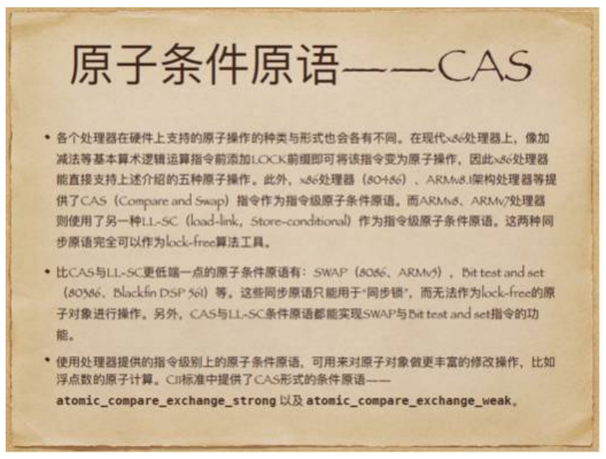


我们要把一个原子对象的值加载到一个普通的，即非原子对象中，我们不能直接用=操作符，而是应该用下面的atomic_load函数，它是将原子对象复赋值给一个普通的变量，如果要把一个普通的变量存储到一个原子对象当中去，我们也是用atomic_store，将一个普通变量的值存储到原子对象当中去。另外，对于atomic_store，我们对原子对象初始化的时候不要用这个，应该用前面提到的方法初始化。

讲好了加载与存储以后，这里简单介绍一下基本的算术逻辑操作。

C11标准是规定了五种可用以原子对象的算术逻辑操作，加、减、或、异或、与，在使用这五个操作的时候，也就是操作数类型不能是atomic_bool类型，对这种类型不支持。这五种操作运算对应到C11原子操作接口，都是属于宏函数，因为每一个编译器实现都会不太一样，分别是atomic_fetch作为前缀，add就是加法，sub就是减法，or就是或，xor就是异或，and就是与。比如：我假定原子对象初始化为100，结果加10，原子对象本身是110没有问题，但是我本身函数返回的值是100，也就是我在加这个原子对象之前的原子对象的值，100，这里大家要当心，这个实现其实是有好处的。

下面来讲一个高级的！因为上面的加法减法属于比较上层的接口，原子操作就是采用无锁，也就是我们要实现无锁数据同步算法的时候，最最本质就是要使用的一个东西就是原子条件原语（CAS）。



C11支持了一个CAS，各个硬件上支持的原子操作类型和形式会不同，X86处理器上，像上文提到的五种操作类型，都能够直接支持，非常简单，只要在这些指令之前加上前缀look就可以，很多处理器是不直接支持原子加法原子减法操作，但是这个时候我们可以通过更底层，更根源的原子操作指令实现。

比如x86处理器以及ARMv8.1架构等处理器直接提供了CAS指令作为原子条件原语。而ARMv7、ARMv8处理器则使用了另一种LL-SC，这两种原子条件原语都可以作为Lock-free算法工具。比如8086时代就有我们大家用的微机原理实验，大家可能会用到一个XCHG，这条指令本身是具有原子性的，也就是我这边写的交换指令SWAP。DSP用的比较多的就是Bit test and set，这些条件原语比起CAS与LL_SC要低档一些，只能用于同步锁。比如我要实现多核多处理器环境下的我们可以用这些原子操作进行，但是他们本身是没有办法作为一个Lock-free的原子对象进行操作。在C11标准当中，就只提供的CAS这种，宏函数接口名为atomic_compare_exchange_strong以及atomic_compare_exchange_weak，第一种是保证数据比较交换是成功还是失败，结果马上就会出来，而这个weak往往针对通过LL-SC指令模拟CAS，里面会产生一些副作用，我做一次比较和交换的时候，我这个结果确实已经交换成功了，但是返回结果可能是失败的，当然我们也可通过一次循环再一次迭代，然后直到它成功返回为止。

那么我们再介绍宏函数的时候，我以strong为例，函数原形是这个样子，返回bool类形，这个函数的语义就是我先比较object的原子对象指针所指的原子对象，与expected所指的内容对象是否相同，如果这两个指针所指的内容相同，我将desired的值存储到object，并且最终返回，我这次修改操作是成功的。否则，也就是expected和object两个内容不相同，这个时候会将object所指的值复制到expected，并且返回true为，我们使用接口的时候我们的操作步序往往是先将atomic原子对象指针的值先拿出来，放到一个普通的变量当中去，我们再去写我object原子对象值的时候我们要用desired，写进去的时候，我先比较expercted为和object是否相同，如果相同就说明我在做原子，从加载到做的过程当中外部没有干扰，也就是我没有存在另外一个线程也使用原子操作，对我当前的object对象进行修改，这个时候两个内容是完全相同的，这个时候显示成功。如果我先用desired对oject的值进行修改的时，这个值被其他线程修改了，也就是我在做atomic_load与atomic_compare_exchange_strong之间有一个缝隙，正好被另外一个线程抓住把柄，它在

当前线程执行atomic_compare_exchange_strong前先修改了object的值，这样就会出现两个值不同，这个时候就会返回false。

下面我就现场给大家大家展示一些DEMO实例（此处略）。

DEMO下载：http://share.weiyun.com/762d2bd48c1edc4f187****52fe7c9bc80