特立独行MVP

【第三章：BAT等名企面试真题解析8讲】第9节：阿里面试真题解析之互斥锁和自旋锁相关问题 已购

来自【《收割BAT：C++校招学习路线总结》】 | 85 浏览 | 0 回复 | 2019-11-14

特立独行MVP 专栏作者 +关注

前言

在多处理器系统环境中需要保护资源避免由于并发带来的资源访问竞争导致的问题，就需要互斥访问，也就是需要引入锁的机制。只有获取了锁的进程才能访问资源。互斥锁和自旋锁是两种代表性的锁。在实际面试当中对于锁相关的问题出现的频率较高，一般涉及到并发访问，线程安全，线程同步相关问题就会问到互斥锁和自旋锁。本文将先对互斥锁和自旋锁的作用，实现以及使用场景做一个重点分析，然后例举6道我在面试阿里时的真题进行分析。

多线程并发访问问题

当多个线程并发访问共享资源时，有可能产生并发访问的安全性问题，可能会导致共享资源被破坏，导致非预期的结果。比如C++ STL当中的vector,map等等都是非并发安全的容器。如果想要解决并发访问的安全性问题就需要引入线程同步机制。

线程间同步指的是：当有一个线程在对共享资源进行操作时，其他线程都不可以对这个资源进行操作，直到该线程完成操作。简单来说，就是线程之间需要达到协同一致。

一般线程间同步机制有：共享内存，信号量机制，锁机制，信号机制等等。其中对于锁的使用是最普遍的方式。

互斥锁：

互斥锁的作用：

互斥锁是为实现保护共享资源而提出一种锁机制。采用互斥锁保护临界区，防止竞争条件出现。当某个线程无法获取互斥锁时，该线程会被挂起，当其他线程释放互斥锁后，操作系统会唤醒被挂起在这个锁上的线程，让其运行。

互斥锁的实现：

在Linux下互斥锁的实现是通过futex这个基础组件。

特立独行MVP

通过用户空间的共享内存以及原子操作，在共享的资源不存在竞争的时候，不会进行系统调用而是只有当竞争出现的情况下再进行系统调用陷入内核。进程或者线程在没有竞争的情况下可以立刻获取锁。具体来说，futex的优化方式如下：

futex将同步过程分为两个部分，一部分由内核完成，一部分由用户态完成；如果同步时没有竞争发生，那么完全在用户态处理；否则，进入内核态进行处理。**减少系统调用的次数，来提高系统的性能是一种合理的优化方式。**

互斥锁的使用场景：

1. 解决线程安全问题，一次只能一个线程访问被保护的资源。
2. 被保护资源需要睡眠，那么可以使用互斥锁。

自旋锁：

自旋锁的作用：

自旋锁也是为实现保护共享资源而提出一种锁机制。自旋锁不会引起调用线程阻塞，如果自旋锁已经被别的线程持有，调用线程就一直循环检测是否该自旋锁已经被释放。

自旋锁的特点：

1. 线程不会阻塞，不会在内核态和用户态之间进行切换。
2. 消耗 CPU: 因为自旋锁会不断的去检测是否可以获得锁，会一直处于这样的循环当中，这个逻辑的处理过程消耗的 CPU相对其实际功能来说是浪费的。

自旋锁的实现：

CAS(compare and swap) 是实现自旋锁的基础。CAS 的实现基于硬件平台的指令。

CAS涉及到三个操作数：

- 需要读写的内存值 value1
- 进行比较的值 value2
- 拟写入的新值 value3

当且仅当 value1 的值等于 value2时，CAS通过原子方式用新值value3来更新value1的值，否则不会执行任何操作。可以理解为线程会不停的执行一个while循环进行CAS操作，直到达成条件。

自旋锁的使用场景：

2. 如果代码当中经常需要加锁但是实际情况下产生竞争的情况比较少此时可以使用自旋锁进行优化。
3. 被保护的共享资源需要在中断上下文访问，就必须使用自旋锁。

原子操作

原子操作的作用：

原子操作是不可被中断的一个或者一系列操作，原子操作可以避免操作被进程/线程的调度打断，原子操作的过程当中不会出现上下文的切换，保证操作的完整性。同时在多处理器的环境下，原子操作也保证了多处理器之间对内存访问的原子性。

原子操作的实现：

原子操作主要是通过硬件操作的方式实现。**在x86平台上，CPU提供了在指令执行期间对**总线加锁的手段，通过将总线锁住，保证其他CPU无法在同一时刻操作内存，从而保证操作的原子性。同时由于缓存的存在，原子操作也需要缓存锁来提供复杂内存情况下的实现。

阿里面试真题

问题1：你知道哪些锁？

答：互斥锁，自旋锁，读写锁，行锁，表锁，乐观锁，悲观锁。

互斥锁：实现互斥操作最简单的方案

自旋锁：无锁操作，比较耗费CPU

读写锁：适合读多写少的场景

行锁：数据库当中细粒度的一种锁实现，只锁一行数据，锁粒度相对较低

表锁：数据库当中粗粒度的一种锁实现，锁整张表，锁粒度较大

乐观锁：当线程去获取数据的时候，乐观地认为别的线程不会修改数据，不对对数据加锁。在更新数据的时候会去通过数据的version(版本号)来判断，如果数据被修改了就拒绝更新。

悲观锁：当线程去获取数据的时候，悲观地以为别的线程会去修改数据，所以线程每次获取数据的时候都会加锁。

解析：在回答有哪些锁的时候，可以同时对每种锁的特点和作用进行介绍；如果对某些锁的实现有比较深入的了解，可以做更多的介绍，比如以某种语言下某种锁的具体实现方式，特点以及优缺点等等。

答：锁的作用是为了控制并发访问的安全性。

解析：为了避免由于并发带来的资源访问竞争导致的问题，就需要互斥访问，所以需要引入锁机制。

问题3：自旋锁和互斥锁的使用场景的区别是什么？

答：互斥锁使用场景：被保护资源需要睡眠，那么只能使用互斥锁或者信号量，不能使用自旋锁。

自旋锁使用场景：锁的持有时间非常短或者被保护的共享资源需要在中断上下文访问。

问题4：如何提升并发访问当中锁的性能？

答：有以下几种方案：

1. 减小锁的粒度：比如锁分段技术
2. 减少锁持有的时间
3. 可以使用自旋锁或者原子操作优化使用互斥锁的地方
4. 读多写少的情况下可以使用特定功能的锁比如读写锁优化互斥锁
5. 读写分离，对读和写的操作采用分离的方式实现

解析：一般在问到Java当中HashMap的实现，Golang当中Sync.Map的实现时，就是对并发访问当中提升锁的性能方式的考察。

问题5：分布式场景下一般使用什么样的锁？

答：可以使用分布式锁，比如使用ETCD来做分布式锁。

解析：分布式锁一般使用版本号和watch机制去实现。

问题6：互斥锁的开销有哪些？

答： 线程(进程)在申请锁时，从用户态切换到内核态，申请到锁之后从内核态返回用户态，这个过程会产生两次上下文切换；线程(进程)在使用完资源后释放锁，从用户态切换到内核态，操作系统会唤醒阻塞等待锁的其他进程，线程(进程)返回用户态，这个过程也会产生两次上下文的切换；而进程上下文切换又包含直接消耗和间接消耗：

1. 直接消耗包括CPU寄存器保存和加载
2. 间接消耗包括TLB的刷新等等

总结

特立独行MVP

结语

感谢阅读，如果文中有任何错误欢迎你给我留言指出，也欢迎分享给更多的朋友一起阅读。

举报

收藏 赞

相关专栏

《收割BAT：C++校招学习路线总结》

19篇文章 | 95订阅

已订阅

0条评论 默认排序

没有回复

请留下你的观点吧~

发布

专栏推荐

《收割BAT：C++校招学习路线总结》

《收割BAT：C++校招学习路线总结》，专栏共计17节。专栏分为五大主要内容，包括后台开发学习...

19篇文章 | 95阅读