多线程同步

小林P251-259

阿秀T24 T18

牛客T13

对于多线程程序来说，同步是指在一定的时间内只允许某一个线程访问某个资源 。 而在此时间内，不允许其他的线程访问该资源 。 可以通过互斥锁（ mutex ）、条件变量（ condition variable ）、读写锁（ reader-writer lock）和信号量（ semphore ）来同步资源 。

互斥锁与自旋锁

互斥锁是一个特殊的变量，它有锁上（ lock ） 和打开 （ unlock）两个状态 。 互斥锁一般被设置成全局变量 。 打开的互斥锁可以由某个线程获得 。 一旦获得，这个互斥锁会锁上，此后只有该线程有权打开，其他想要获得互斥锁的线程， 会等待直到互斥锁再次打开的时候 。

条件变量

条件变量互斥量是线程程序必需的工具，但并非是万能的 。 例如，如果线程正在等待共享数据内某个条件出现，那会发生什么呢？它可能重复对互斥对象锁定和解锁，每次都会检查共享数据结构，以查找某个值 。 但这是在浪费时间和资源，而且这种繁忙查询的效率非常低 。
在每次检查之间，可以让调用线程短暂地进入睡眠，比如睡眠 3 秒，但是由此线程代码就无法最快作出响应 。 真正需要的是这样一种方法： 当线程在等待满足某些条件时使线程进入睡眠状态，一旦条件满足，就唤醒因等待满足特定条件而睡眠的线程 。 如果能够做到这一点，线程代码将是非常高效的，并且不会占用宝贵的互斥对象锁 。 而这正是条件变量能做的事!
条件变量通过允许线程阻塞和等待另一个线程发送信号的方法弥补互斥锁的不足，它常和互斥锁一起使用 。 使用时，条件变量被用来阻塞一个线程，当条件不满足时，线程往往解开相应的互斥锁并等待条件发生变化。 一旦其他的某个线程改变了条件变量，它将通知相应的条件变量唤醒一个或多个正被此条件变量阻塞的线程，这些线程将重新锁定互斥锁并重新测试条件是否满足 。

读写锁

在一些程序中存在读者写者问题，也就是说，对某些资源的访问会存在两种可能的情况， 一种是访问必须是排他性的，就是独占的意思，这称作写操作；另一种情况就是访问方式可以是共享的，就是说可以有多个线程同时去访问某个资源，这种就称作读操作。这个问题模型是从对文件的读写操作中引申出来的。

- 读写锁比起互斥锁具有更高的适用性与并行性，可以有多个线程同时占用读模式的读写锁，但是只能有一个线程占用写模式的读写锁，读写锁的 3 种状态如下所述。
    - 当读写锁是写加锁状态时，在这个锁被解锁之前，所有试图对这个锁加锁的线程都会被阻塞。
    - 当读写锁在读加锁状态时，所有试图以读模式对它进行加锁的线程都可以得到访问权，但是以写模式对它进行加锁的线程将会被阻塞。
    - 当读写锁在读模式的锁状态时，如果有另外的线程试图以写模式加锁，读写锁通常会阻塞随后的读模式锁的请求，这样可以避免读模式锁长期占用，而等待的写模式锁请求则长期阻塞。

> 读写锁最适用于对数据结构的读操作次数多于写操作次数的场合，因为读模式锁定时可以共享 ， 而写模式锁定时只能由某个线程独占资源，因而读写锁也可以叫作共享 - 独占锁 。
> 处理读者 - 写者问题的两种常见策略是强读者同步（strong reader synchronization ）和强写者同步（ strong writer synchronization ） 。 在强读者同步中，总是给读者更高的优先权，只要写者当前没有进行写操作， 读者就可以获得访问权限；而在强写者同步中，则往往将优先权交付给写者，而读者只能等到所有正在等待的或者正在执行的写者结束以后才能执行。 关于读者 - 写者模型， 由 于读者往往会要求查看最新的信息记录 ， 例如航班订票系统往往会使用强写者同步策略，而图书馆查阅系统则采用强读者同步策略。

- 读写锁机制是由 POSIX 提供的，如果写者没有持有读写锁，那么所有的读者都可以持有这把锁，而一旦有某个写者阻塞在上锁的时候，那么就由 POSIX 系统来决定是否允许读者获取该锁 。

悲观锁和乐观锁

信号量

线程还可以通过信号量来实现通信。信号量和互斥锁的区别： 互斥锁只允许一个线程进入临界区，而信号量允许多个线程同时进入临界区 。