6.3.1 优先级倒置

很多实时系统都实现了线程优先级机制，每个线程被分配一个优先级，优先级高的线程只要具备了运行的条件，或者说进入了就绪态，就可以立即执行。除了优先级最高的那个线程外，其他线程在执行过程中随时都可能停下来，让给优先级更高的线程先执行。抢占式调度策略能保证系统的实时性。

由于多个线程共享资源，在采用基于优先级调度策略时会出现较低优先级的线程先于高优先级线程执行的情况，即优先级倒置（priority inversion）问题。优先级倒置可能使实时性要求高的线程错过临界期限(critical deadline)，从而导致系统崩溃。

在采用基于优先级的调度策略，一旦两个线程共享了资源，那么其中之一通常具有较高的优先级。高优先级的线程期望一旦准备就绪就能运行，但当高优先级线程就绪，而低优先级线程正在使用共享资源时，高优先级线程必须等待，直到低优先级线程完成对共享资源的操作。这时，我们称高优先级线程被挂起。当高优先级线程挂起时，中优先级的线程抢先了正在使用共享资源的低优先级线程，此时高优先级线程已准备就绪并等待运行，但中优先级线程此刻正在运行，这时就出现了优先级倒置问题。

下图以三个不同优先级的线程为例描述一个优先级倒置的示例。高优先级线程 H 和低优先级线程 L 要共享资源 R，为了保证数据的完整性，它们需要通过信号量 S 来保证对临界资源的互斥访问。线程 M 的优先级介于 H 和 L 之间。

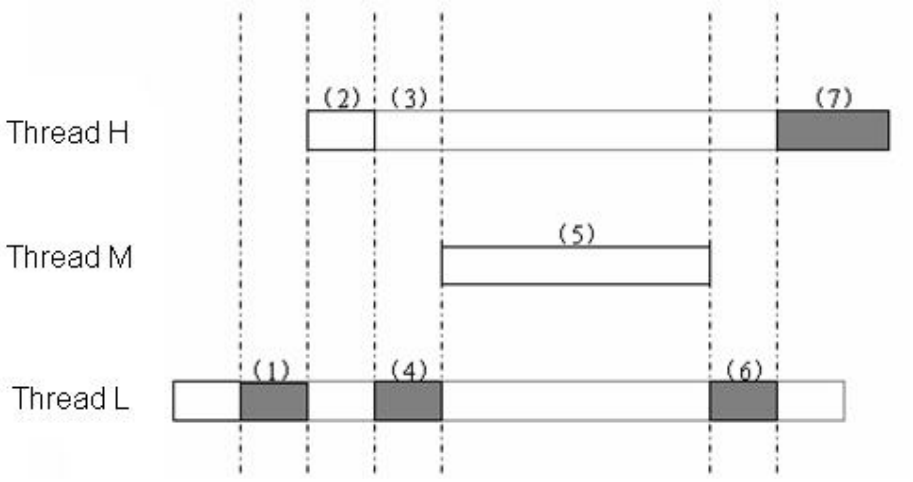

图 6.1 一个优先级倒置的示例

（1）低优先级线程 L 取得信号量 S 的所有权，即做了 P 操作，但还没有做 V 操作。

（2）线程 H 的优先级高于线程 L，操作系统内核通过调度程序，将线程 L 切换出去，将线程 H 置为运行态。

（3）线程 H 执行到中途需要访问共享资源 R，必须先对信号量 S 做 P 操作。因为此信号量目前还没有恢复，于是线程 H 阻塞在信号量 S 上。

（4）线程 L 重新被切换到运行态。

（5）此时线程 M 进入到就绪态，因为线程 M 的优先级高于任务 L，于是内核进行线程切换，将线程 M 置为运行态。

（6）线程 M 执行结束后，线程 L 恢复执行。

（7）线程 L 释放共享资源 R，即对信号量做 V 操作。线程 H 获得信号量 S 的所有权才得以继续执行。

从上面的分析可以看到，由于高优先级线程 H 要获取被低优先级线程 L 占有的临界资源而被阻塞，具有中等优先级的线程 M 抢占线程 L 的 CPU，从而导致线程 M 先于线程 H 执行。这时便产生了优先级倒置的情况。

要避免出现优先级倒置问题，必须让低优先级线程尽快释放临界资源。目前解决优先级倒置通常有两种方法，一种是优先级继承（priority inheritance），另一种是优先级顶置（priority ceilings）。

优先级继承

优先级继承技术强令低优先级的线程继承与之共享资源并被挂起的高优先级线程的优先级。一旦高优先级线程开始挂起，即可实施优先级继承，直到资源释放。下图是一个优先级继承的示例。

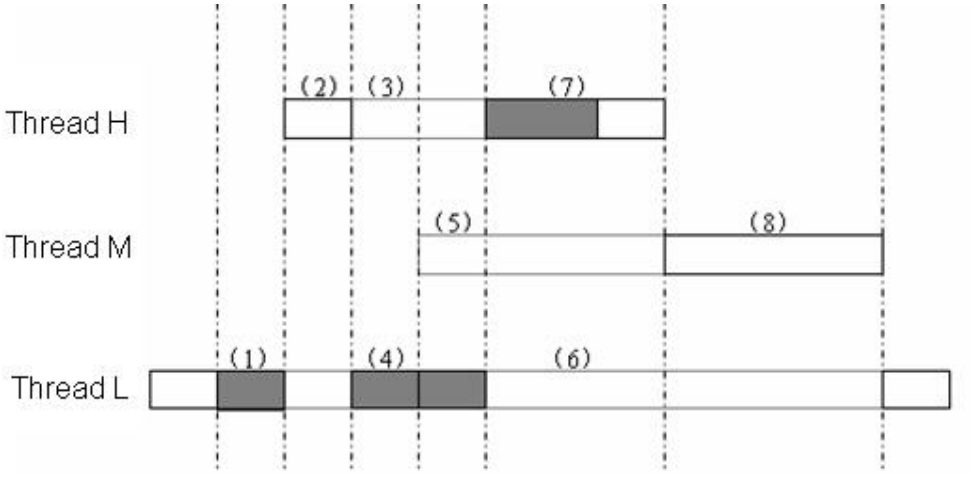

图 6.2 优先级倒置的产生

（1）低优先级线程 L 取得信号量 S 的所有权，即做了 P 操作，但还没有做 V 操作。

（2）线程 H 的优先级高于线程 L，操作系统内核通过调度程序，将线程 L 切换下去，将线程 H 置为运行态。

（3）线程 H 执行到中途需要访问共享资源 R，必须先对信号量 S 做 P 操作。因为此信号量

目前还没有恢复，于是线程 H 阻塞在信号量 S 上。

（4）线程 L 重新被切换到运行态，并继承线程 H 的优先级。

（5）此时线程 M 进入到就绪态，因为线程 M 的优先级低于线程 L 此时的优先级，因此线程 L 继续执行。

（6）线程 L 释放共享资源 R，即对信号量做 V 操作。线程 L 恢复其原来的优先级。

（7）线程 H 的优先级最高，于是内核进行任务切换，将线程 H 置为运行态。

（8）线程 H 执行结束后，按优先级高低顺序依次调度线程 M 和线程 L。

在优先级继承方案中，当高优先级线程在等待低优先级的线程占有的临界资源时，让低优先级线程继承高优先级线程的优先级，即把低优先级线程的优先级提高到高优先级线程的优先级。当低优先级线程释放高优先级线程等待的临界资源时，立即把其优先级降低到原来的优先级。采用这种方法可以有效地解决优先权倒置的问题。这种方法的缺点是，动态改变线程的优先级耗费了大量的时间。

优先级顶置

优先级顶置方案为每个临界资源都分配一个优先级。假设线程 H 在所有要共享某资源的线程中优先级最高，则将资源的优先级确定为线程 H 的优先级加 1。调度程序将该资源的优先级赋给任何访问该资源的线程，这样就能保证该线程能尽快完成对临界资源的访问。一旦线程完成对该资源的操作，其优先级恢复正常。下图是一个优先级顶置的示例。

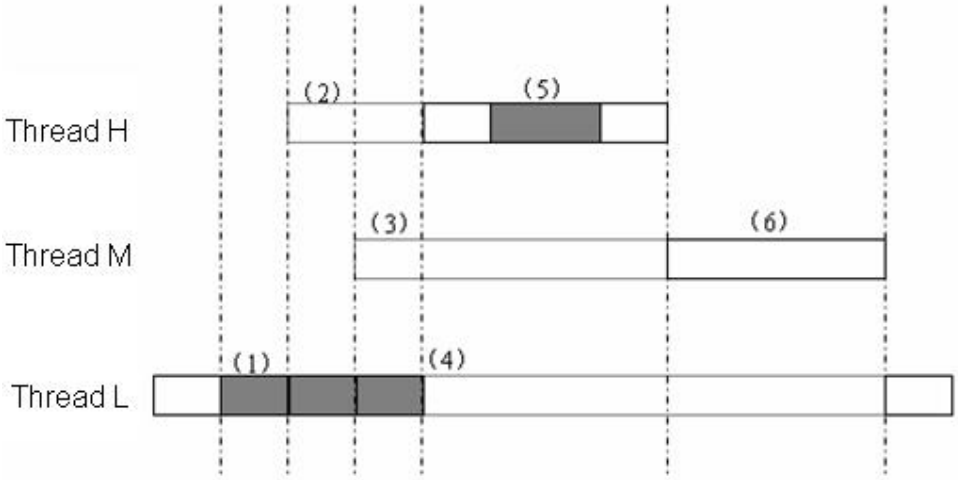

图 6.3 优先级顶置

（1）低优先级线程 L 取得信号量 S 的所有权，即做了 P 操作，但还没有做 V 操作；线程 L 优先级被置为 H+1。

（2）线程 H 进入到就绪态，因为线程 H 的优先级低于线程 L 此时的优先级，因此线程 L 继续执行。

（3）线程 M 进入到就绪态，因为线程 M 的优先级低于线程 L 此时的优先级，因此线程 L 继续执行。

（4）线程 L 释放共享资源 R，即对信号量做 V 操作。线程 L 恢复其原来的优先级。

（5）线程 H 的优先级最高，于是内核进行任务切换，将线程 H 置为运行态。

（6）线程 H 执行结束后，按优先级高低顺序依次调度线程 M 和线程 L。

这种方法目的是让低优先级线程尽快释放临界资源。假如没有线程 H 存在，但该方法却推迟了线程 M 的执行。这种情况在优先级继承方法中是不会出现的。

优先级顶置方法总是乐于提高线程的优先级，而优先级继承方法则比较懒惰，不到万不得已不会提升线程的优先级。

对于使用锁或者忙等待机制的编程人员来说，如果有不同优先级的线程被允许获取锁，就会遇到优先级倒置问题。手工编码的旋转锁就是这样一种很常见的技术。如果在锁或忙等待机制中无法采用优先级继承或优先级顶置技术，那么就最好将参与锁竞争的线程限制为具有同样的优先级。