3.4 本章总结

本章讨论了当两个线程间的共享数据发生恶性条件竞争会带来多么严重的灾难，还讨论了如何使用 `std::mutex` ，和如何避免这些问题。如你所见，互斥量并不是灵丹妙药，其还有自己的问题(比如：死锁)，虽然C++标准库提供了一类工具来避免这些(例如： `std::lock()`)。你还见识了一些用于避免死锁的先进技术，之后了解了锁所有权的转移，以及一些围绕如何选取适当粒度锁产生的问题。最后，讨论了在具体情况下，数据保护的替代方案，例如: `std::call_once()` 和 `boost::shared_mutex` 。

还有一个方面没有涉及到，那就是等待其他线程作为输入的情况。我们的线程安全栈，仅是在栈为空时，抛出一个异常，所以当一个线程要等待其他线程向栈压入一个值时(这是一个线程安全栈的主要用途之一)，它不得不多次尝试去弹出一个值，当捕获抛出的异常时，再次进行尝试。这种消耗资源的检查，没有任何意义。并且，不断的检查会影响系统中其他线程的运行，这反而会妨碍程序的进展。我们需要一些方法让一个线程等待其他线程完成任务，但在等待过程中不占用CPU。第4章中，会去建立一些工具，用于保护共享数据，还会介绍一些线程同步操作的机制；第6章中，如何构建更大型的可复用的数据类型。