进行了课件中的多线程实验，创建了20个线程，并结合操作系统的相关知识对结果进行了分析。

使用多线程必须注意一个问题，多个线程之间共享资源。如果有多个线程同时运行，而且它们试图访问相同的资源，就会遇到各种问题。如请求在代码在阅读时写入其他函数，即一个线程要读，一个线程要写，读线程占据着资源，写线程得不到资源，自然就会与理想的结果相违背。

多线程中，变量都由所有线程共享。所以，任何一个变量都可以被任何一个线程修改。如修改一个变量需要多条语句，在执行这几条语句时，线程可能中断，很可能把内容给改乱了。如下所示的代码，创建一个为0的变量，创建两个线程（解释容易），执行+1的操作，得到的最后结果不一定是2。

代码环境：gcc版本: 10.2.0，Manjaro Linux 20.1.1。

代码按以下顺序执行顺序时，便不会出现理想答案。`t1, t2, t3, t4`为多个线程，`x1, x2, x3`为各自线程的临时变量：

```
初始值 balance = 0

t1: x1 = g + 1      # x1 = 0 + 1 = 1
t2: x2 = g + 1      # x2 = 0 + 1 = 1
t3: x3 = g + 1      # x3 = 0 + 1 = 1

t4: x1 = g + 1      # x1 = 1 + 1 = 2
t4: g = x1          # g = 2

t2: g = x1          # g = 1
```

而执行顺序则完全取决于操作系统的调度方案和CPU的核心是否忙碌。如：操作系统是否允许任务长时间占用CPU，CPU在不忙碌的情况下不会进行线程切换，只会执行单个线程。一旦发生线程切换，且没有对临界资源添加保护锁，很容易写出危险的程序。

为了保证变量的正确，务必要给操作函数上一把锁子，当多个线程同时执行时，只有一个线程能成功地获取锁，然后继续执行代码，其他线程就继续等待锁释放，直到获得锁为止。获得锁的线程用完后一定要释放锁，否则那些苦苦等待锁的线程将永远等待下去，成为死线程。

锁的好处就是确保了某段关键代码只能由一个线程从头到尾完整地执行，坏处当然也很多。首先是程序操作临界资源时，阻止了多线程并发执行，包含锁的某段代码实际上只能以单线程模式执行，也降低了效率。临界资源举例：如打印机，同一时刻只能由一个线程控制打印，如写文件，同一时刻只能有一个线程在写。