**Barriers**

一、实验目的

实现一个线程屏障，多个线程都会调用barrier()函数，要求所有的线程都进入此barrier函数后才能依次离开barrier

二、实验内容

[Homework: Barriers (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/barrier.html)

三、实验步骤与实验结论

（1）下载 barrier.c 并编译运行

$ gcc -g -O2 -pthread barrier.c

$ ./a.out 2

(2) 修改 barrier.c 中的函数 barrier()

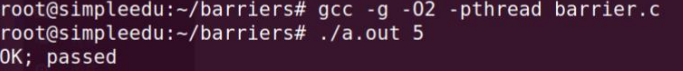
barrier\_init已经实现了，还需实现barrier()，以致于不会再发生 panic

最后一个进程进入执行 round++，再唤醒其他进程。

程序的main函数闯将n个线程，每个线程执行1个for循环，在每次循环时assert检测循环次数是否与全局统计值相等，然后调用barrier函数，最后sleep。实现在barrier函数中确保线程都执行完1次循环才开始下1次循环。

先上锁，增加线程数统计变量barrier.nthread，判断是否已经达到n，如果没有则调用条件变量等待，如果达到了，那么就增加循环次数统计变量barrier.round，同时将barrier.nthread置0，然后广播通知所有线程，唤醒等待中的线程。





注意：是否存在假唤醒的可能？

正常情况下，如果一共是4个线程，那么当前三个线程进入后，他们都会被堵塞在AAA语句中，直到第四个线程进入后执行了BBB语句，前3个线程都会从wait中醒来，然后依次获取锁解锁再退出barrier()。

但是，如果前三个线程堵塞在AAA后，线程4还没有进入barrier()函数，然后此时发生了操作系统层面的假唤醒，导致前3个线程从wait中返回，这时它们就会直接退出barrier()

四、反思总结

必须处理一连串的 barrier call ，

必须处理一个线程在其他线程退出 barrier 之前进行下一轮循环的情况。