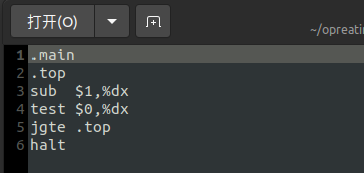
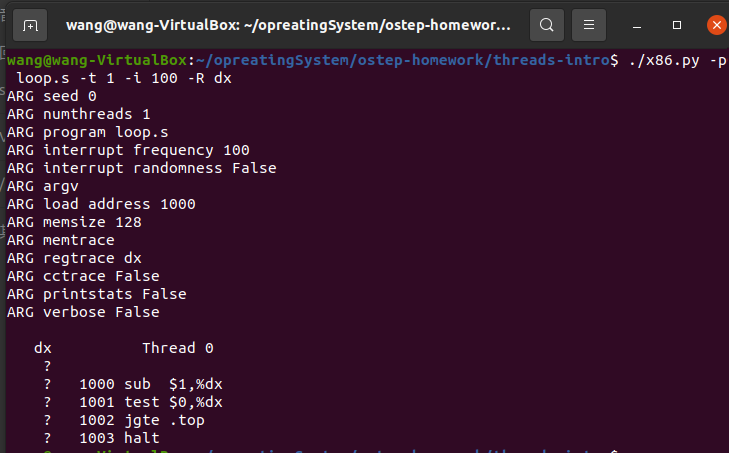
26.1



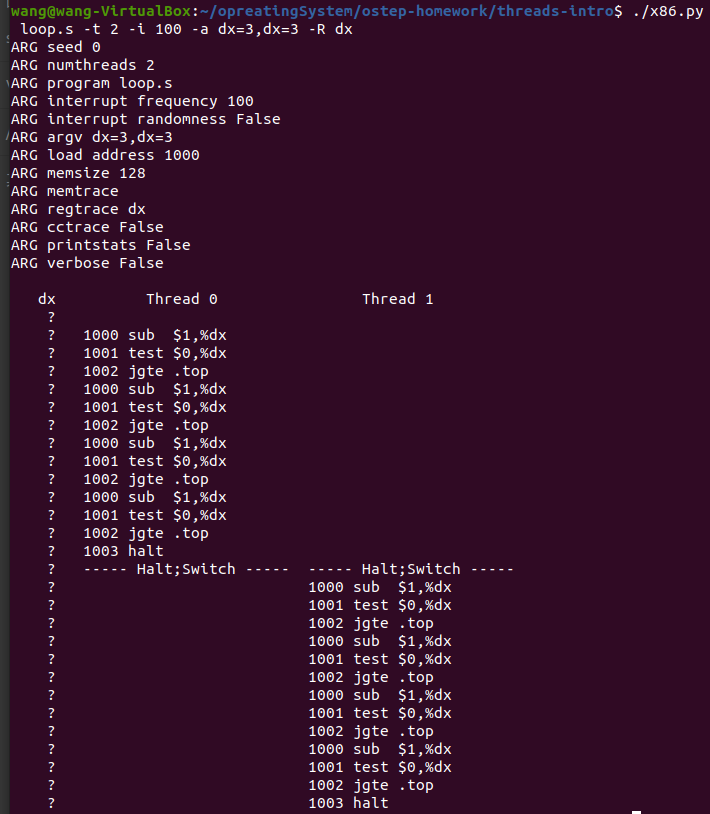


top开始线程，接下来将[寄存器](https://so.csdn.net/so/search?q=%E5%AF%84%E5%AD%98%E5%99%A8&spm=1001.2101.3001.7020)dx的值减一，再用dx中的值和0作比较。如果大于等于则跳转到top. halt是结束这个线程

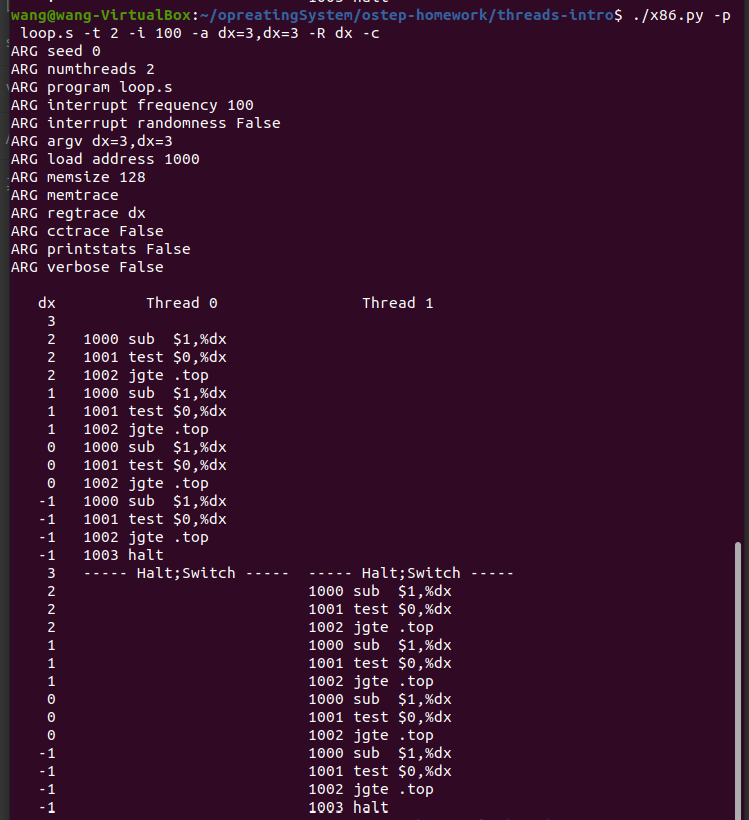
下面是使用-c查看结果



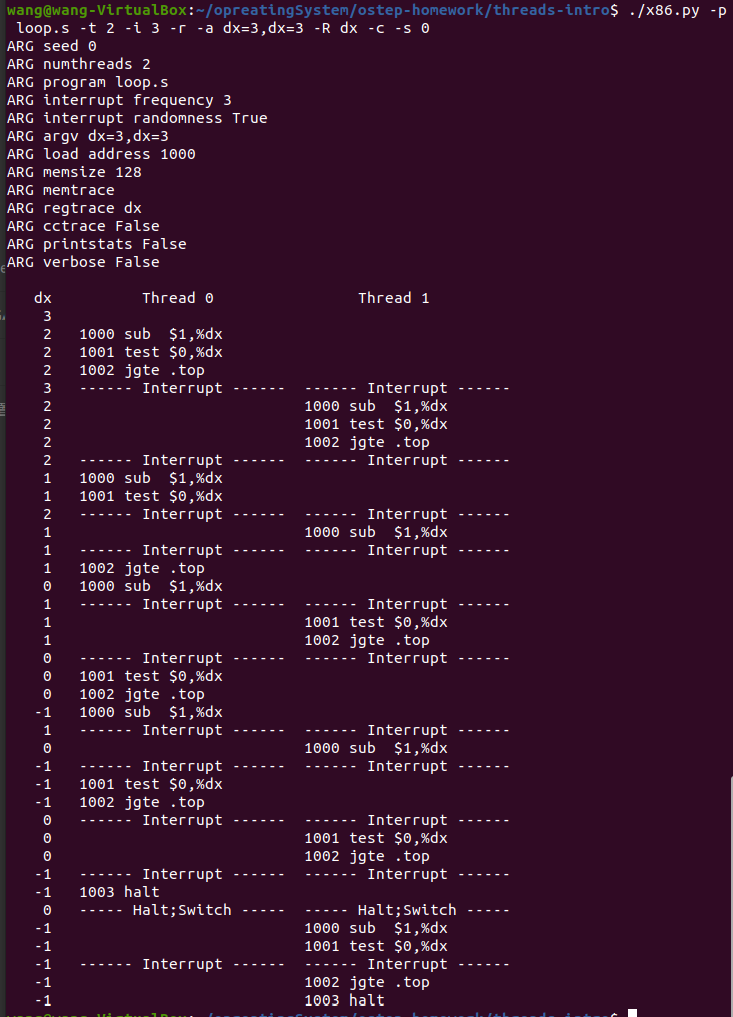
26.2

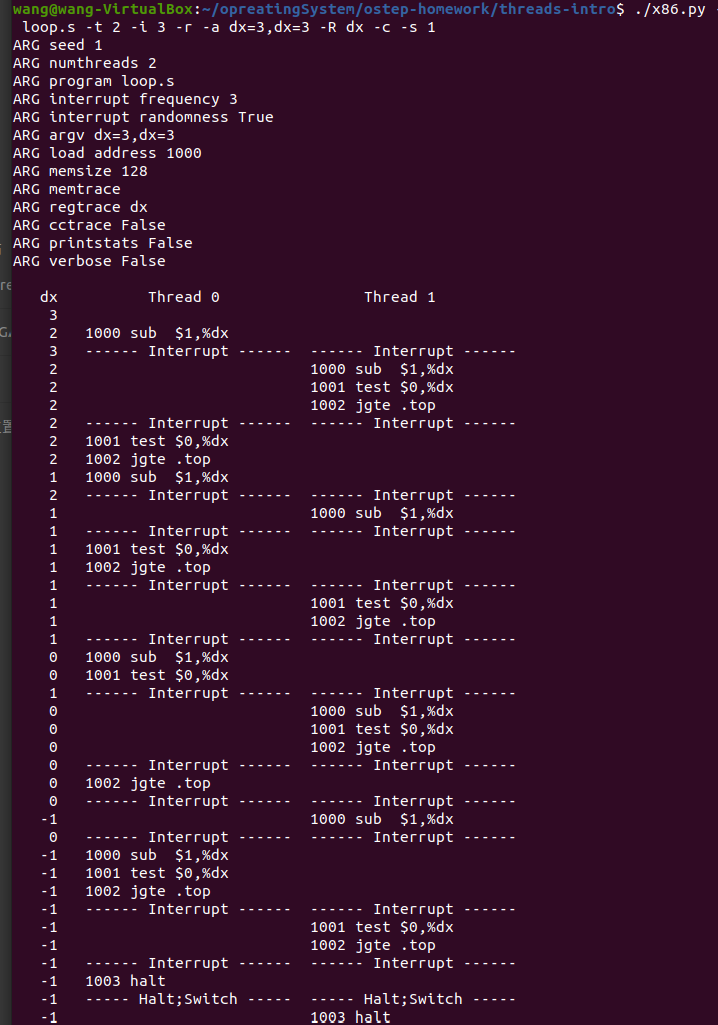


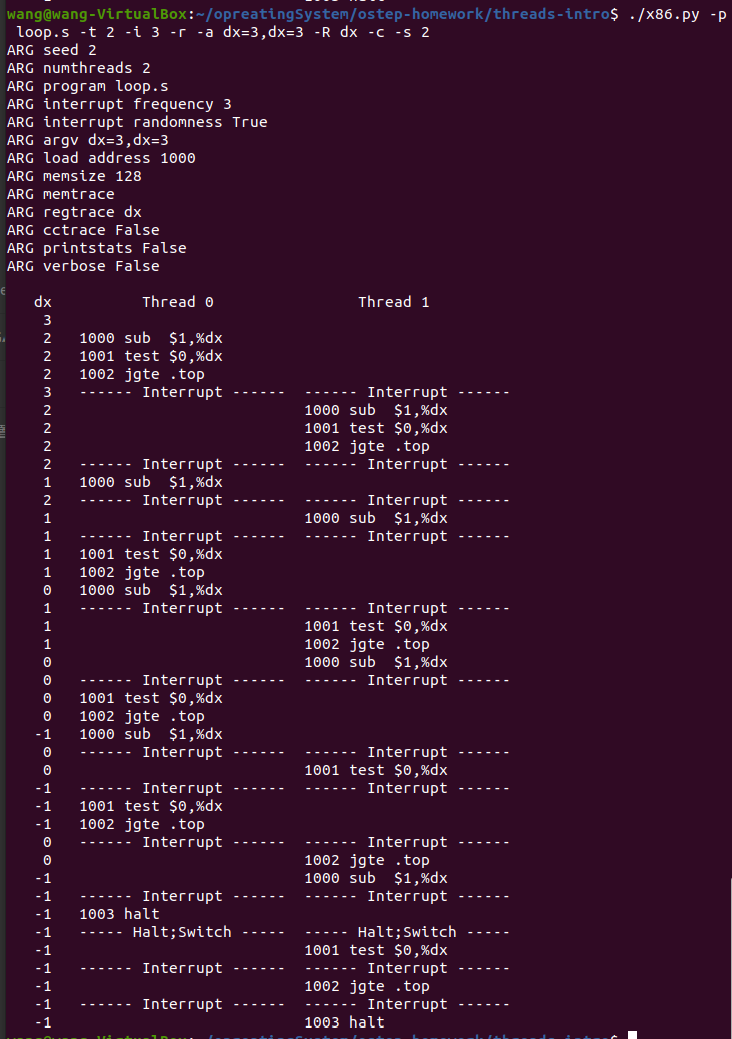
运行完上述过程的所有指令总数没有达到100条，故不会发生中断，使用-c查看结果



26.3

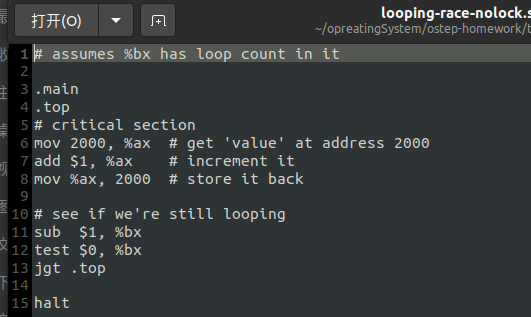


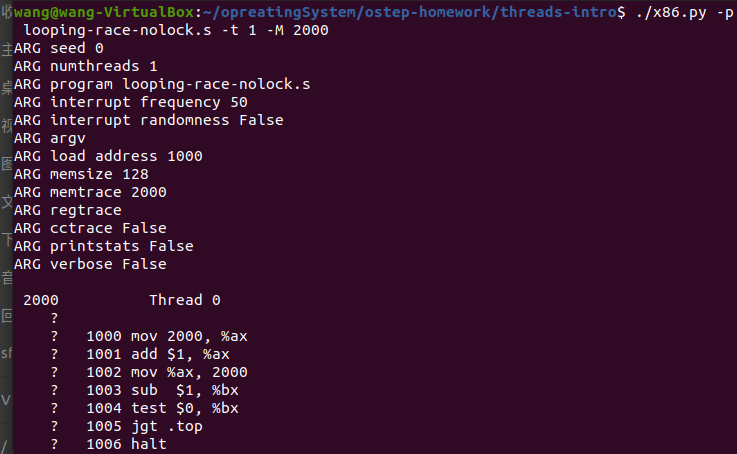




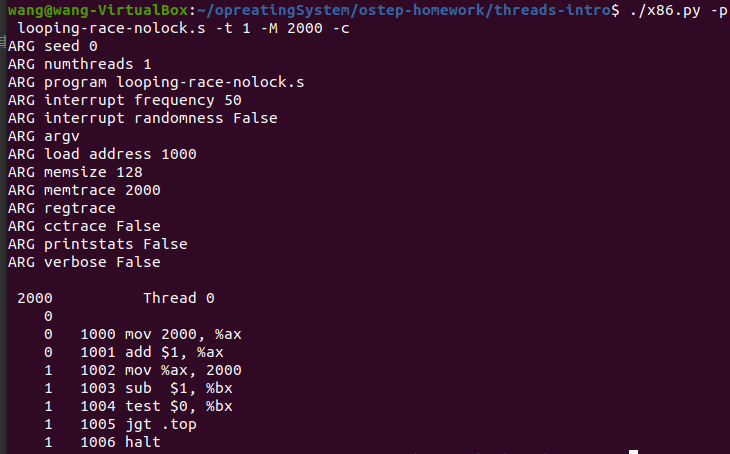
由三个种子的情况，当中断条件变得很短时（一个线程还没有运行完就中断了），中断时就会保存当前状态，等再次回到这个线程时恢复状态并接着工作

26.4

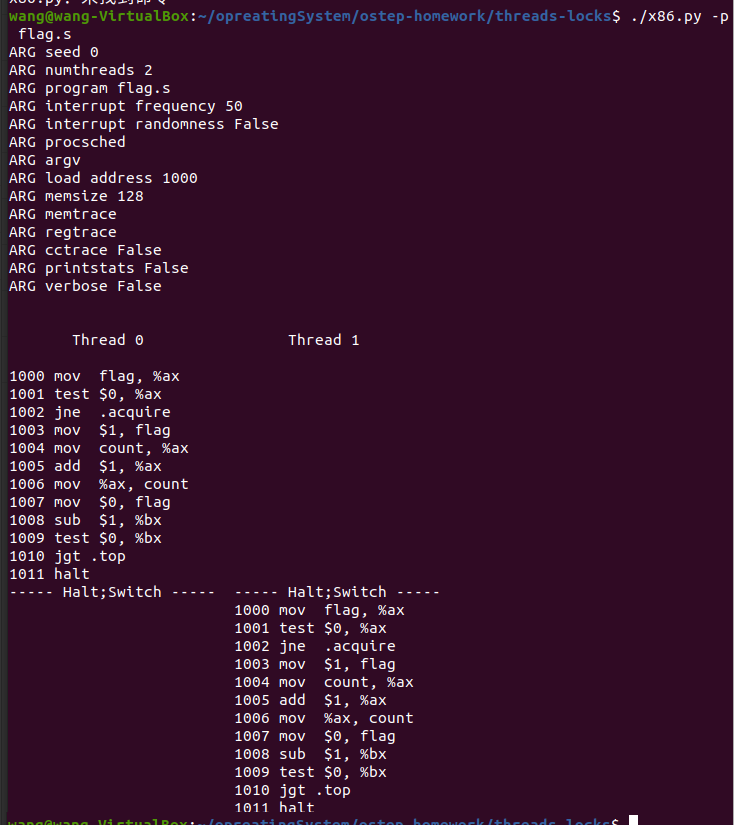


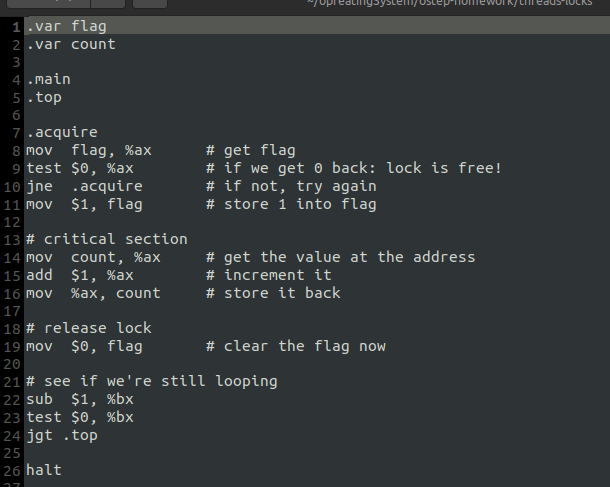


把地址2000中的值给%ax，%ax中的值加一，再把更新后的值给%ax。%bx的值减一，判断%bx和0，如果%bx大于0则跳转到top处，否则就运行halt结束线程



28.1

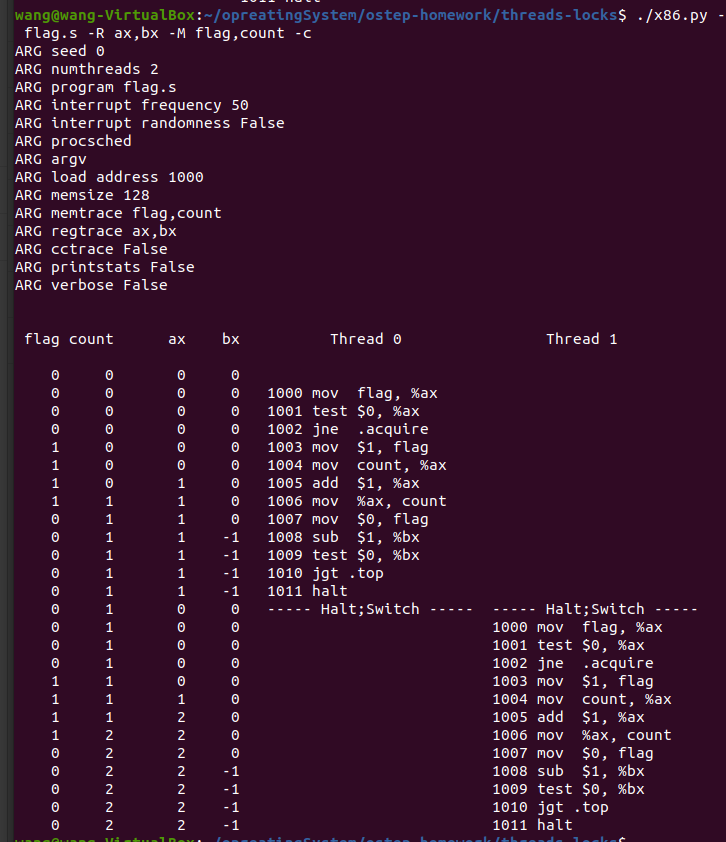




这里用flag储存锁的开关状态，对临界区代码进行保护

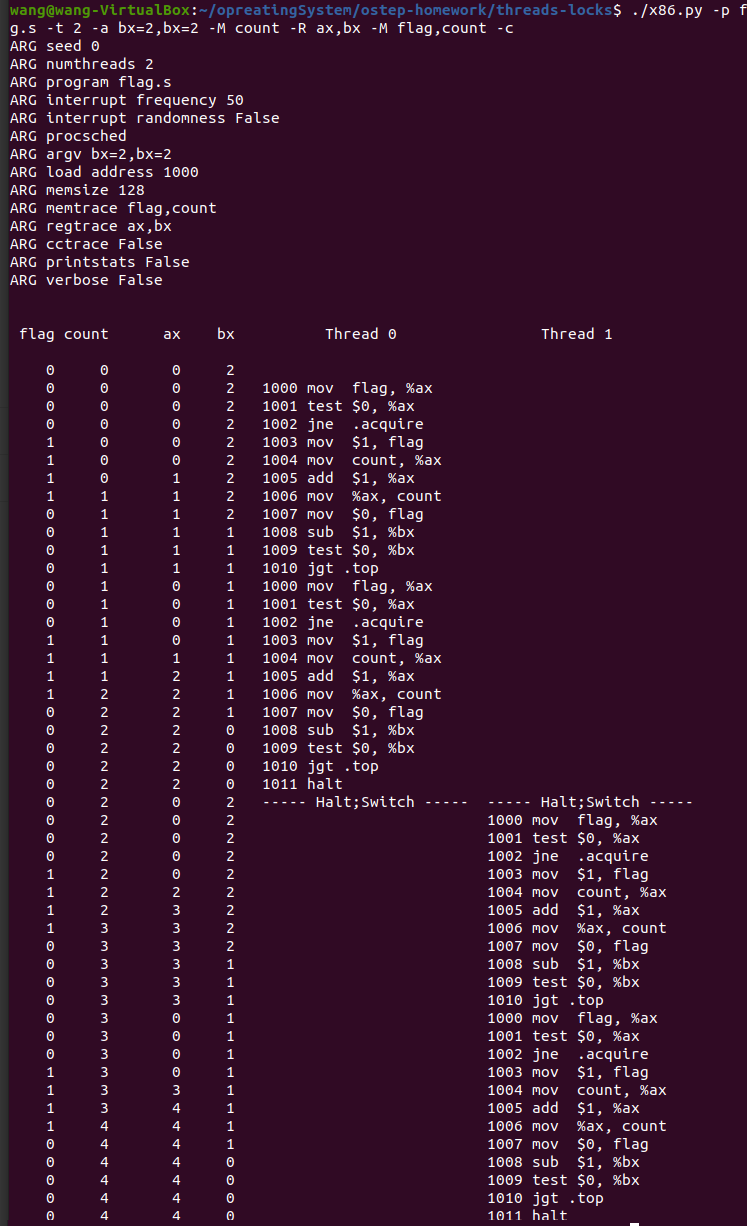
28.2

使用默认值运行时，flags会按预期工作，可以产生预期的正确结果



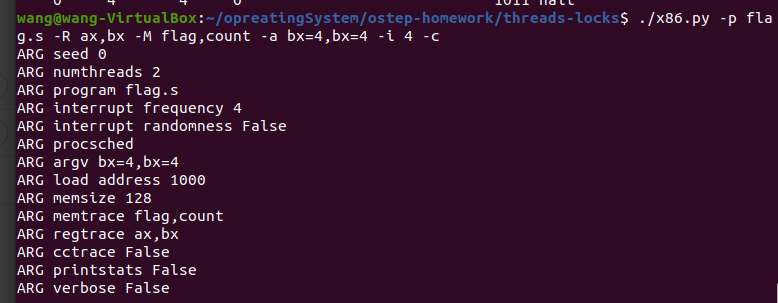
28.3

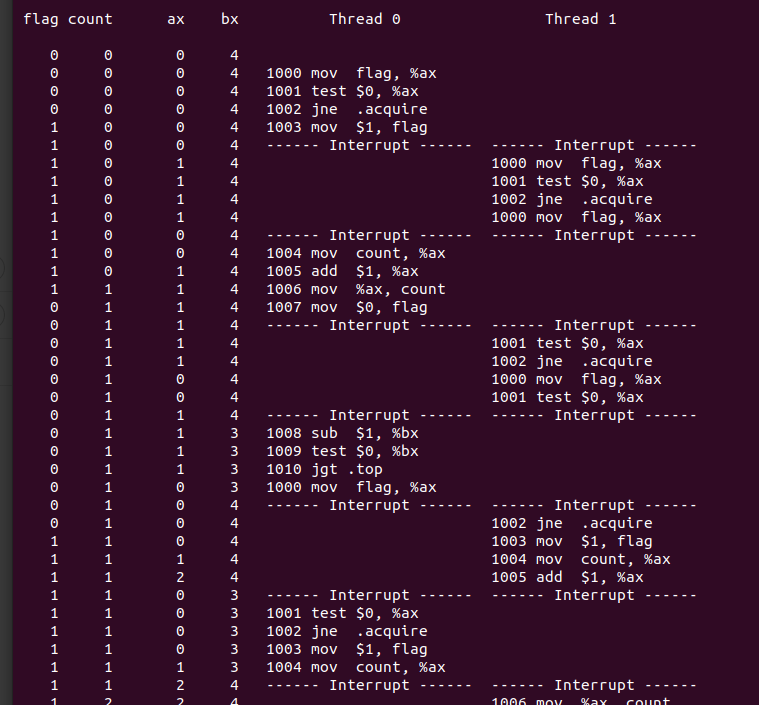
-a 可以更改寄存器%bx的值，将两个线程的寄存器bx值设置为2，运行两个线程， 每个线程都会重复执行两次, 由于两个线程都是完整地进行获得锁，释放锁的过程，所以最终flag=0



28.4

将bx设置为高值，单个线程执行循环的次数增多，指令的数目也会增多

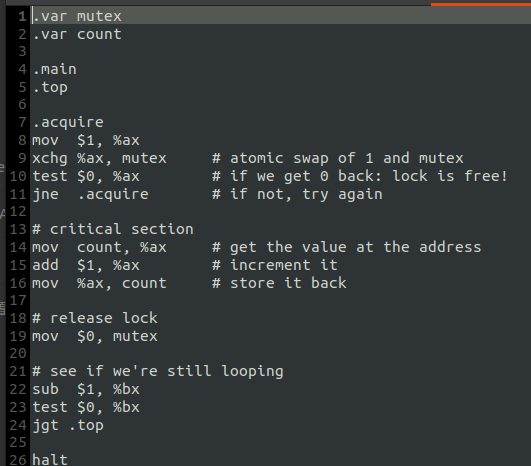




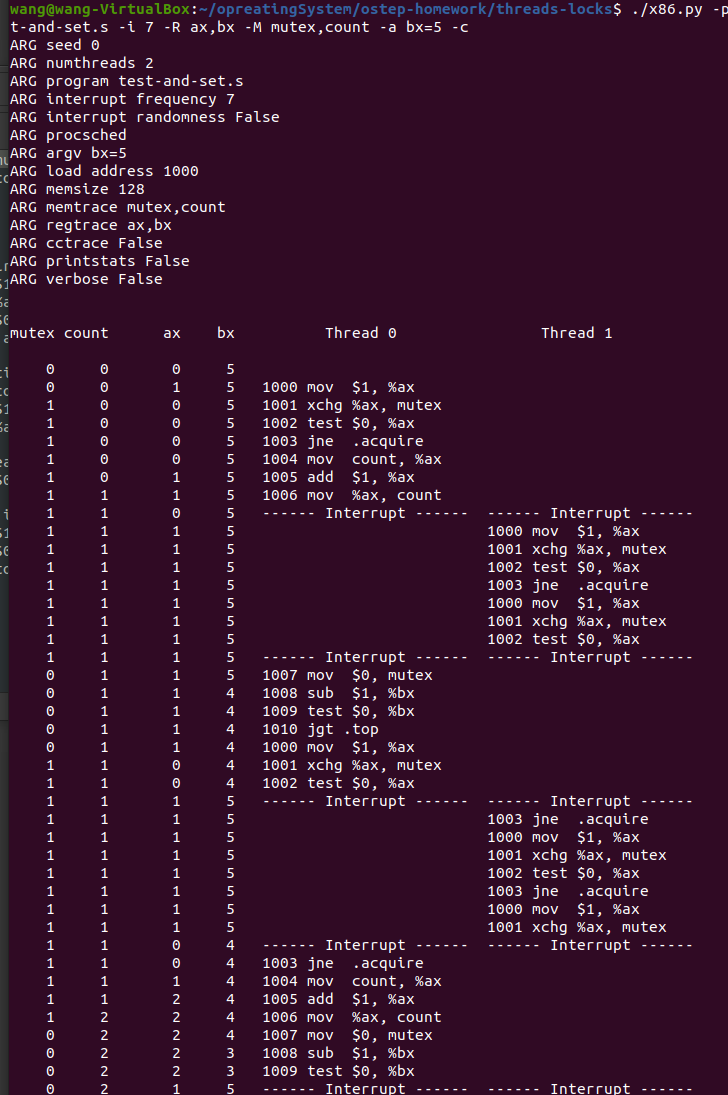
bx为高值说明要进行的循环次数很多，这是如果中断的频率很高，就很容易出问题， i的值越小（中断频率越高）会导致产生不好的结果；i的值越大（中断频率越低）会导致产生好的结果。

28.5

由于取flag和交换ax与flag的值这两个操作是分开进行的，所以在中断 时可能出现同时对flag写入的情况

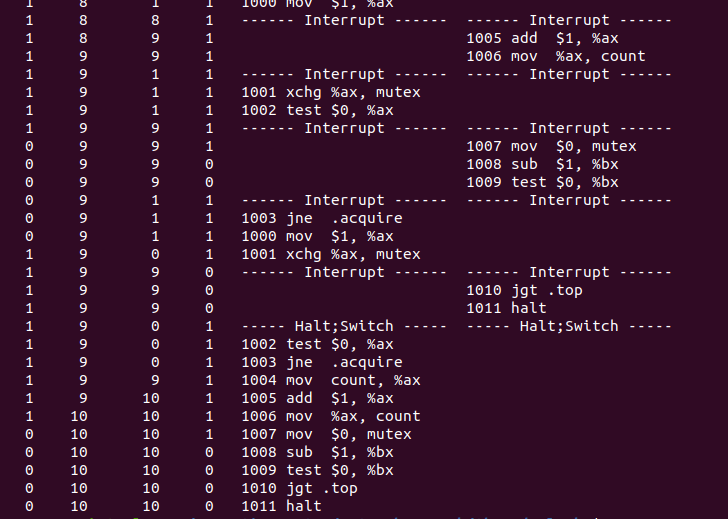


28.6



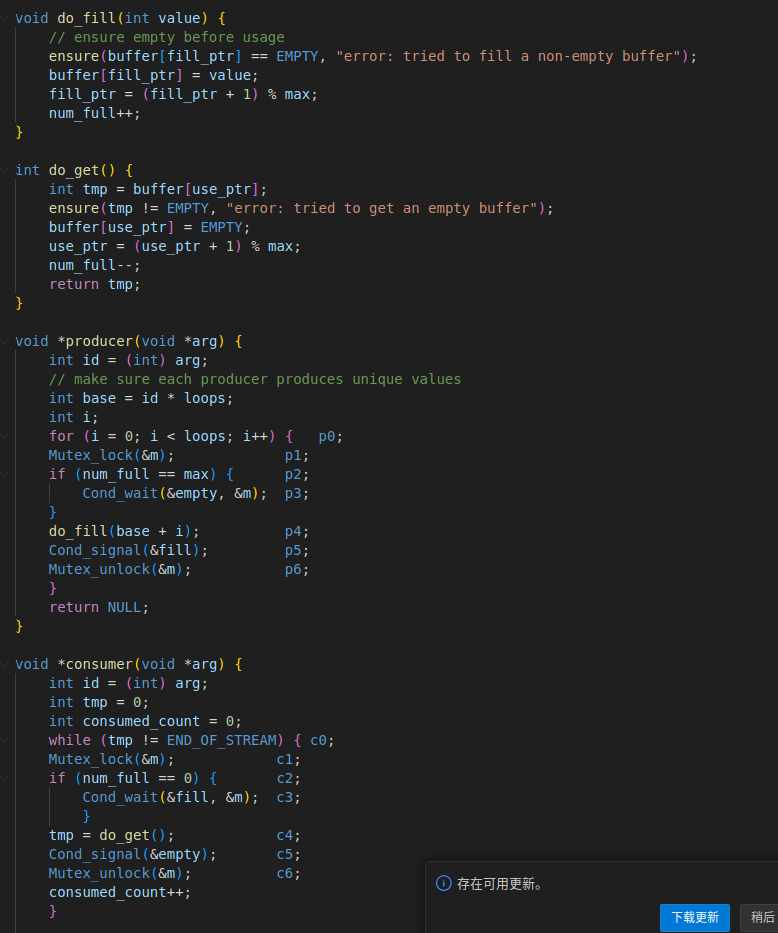
由于xchg的原子操作，我们的锁能够提供互斥，两个线程不会同时进入临界区,代码能够按预期正常工作，但是题4中的问题依然存在，不合时宜的中断会占用cpu却不做实事

28.7



正确发生了

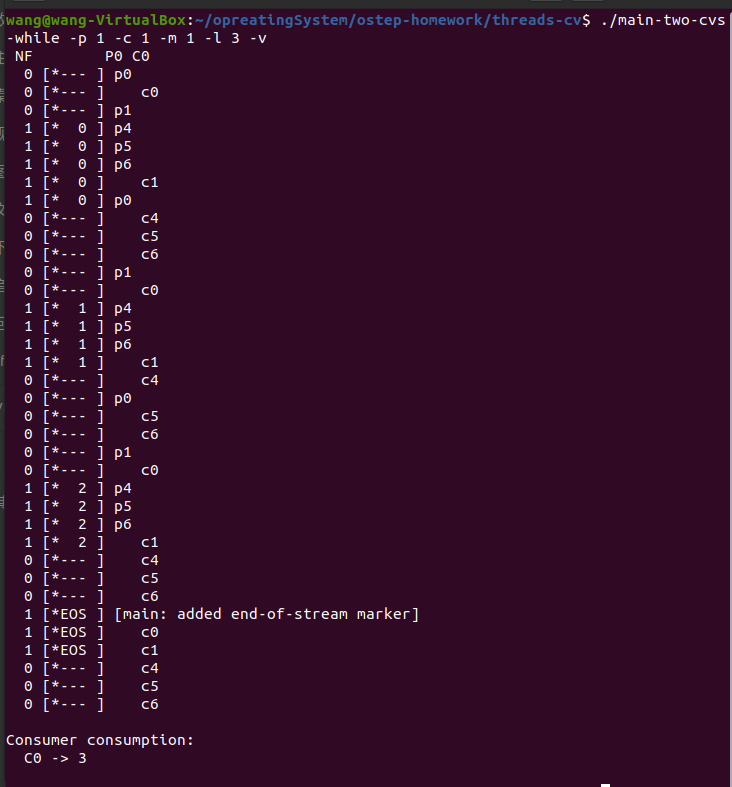
30.1

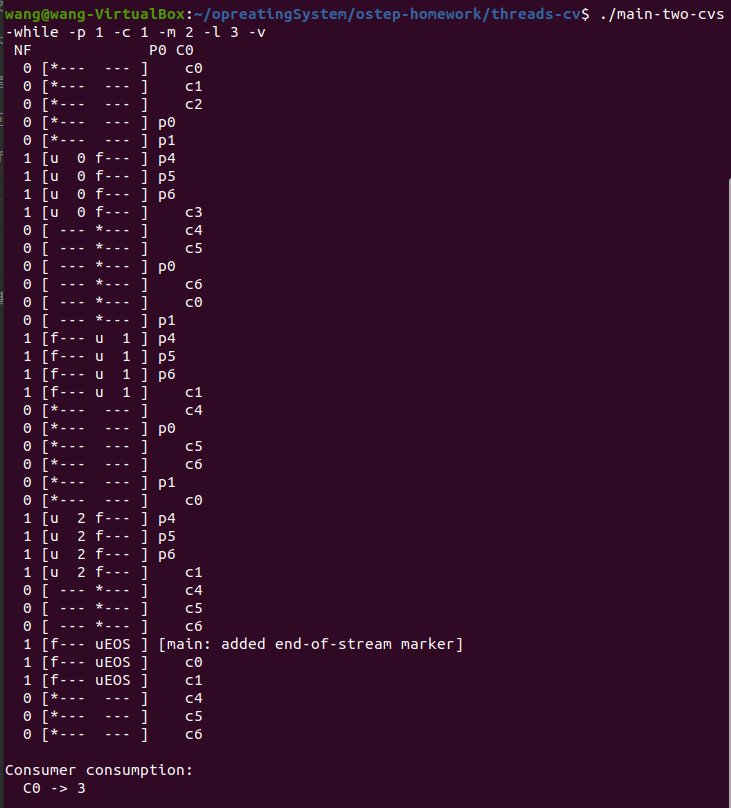


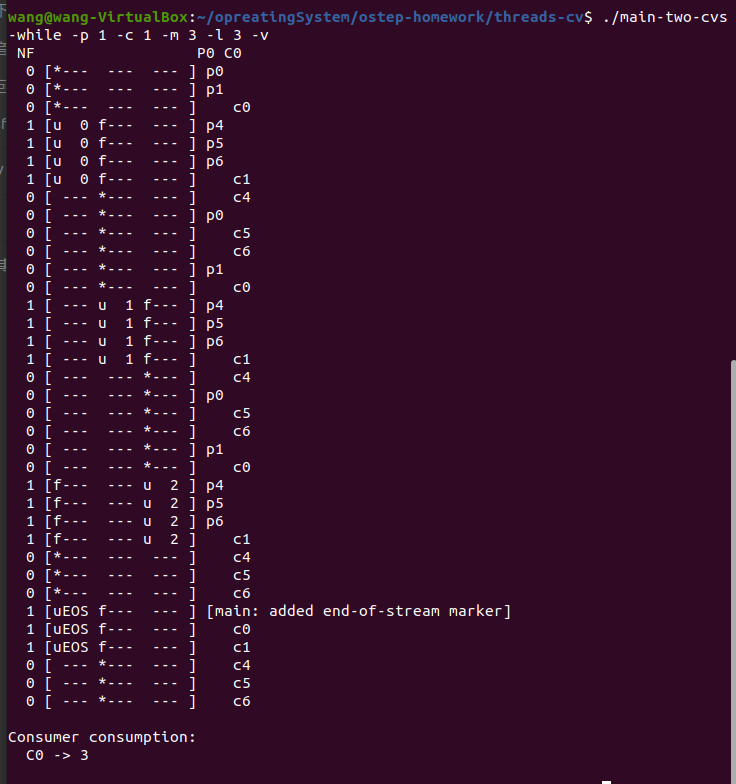
消费过程如下：首先获取一把锁，然后如果缓冲区中没有数据，则等待一个数据信号。否则，取出缓冲区中的数据。并发出一个信号。然后释放锁，并让自己的消费计数器+1

30.2

对项目进行make后

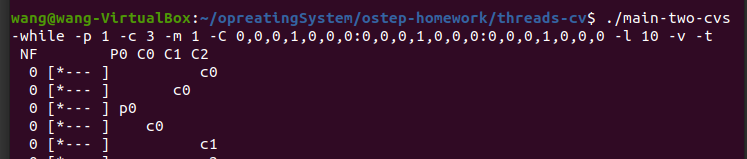


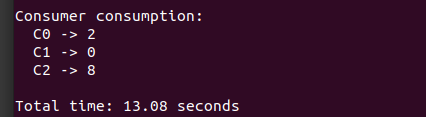




程序运行的结果基本不变，随着m的增加，生产者和消费者每次防止数据的位置和消费的 数据有所不同

30.4



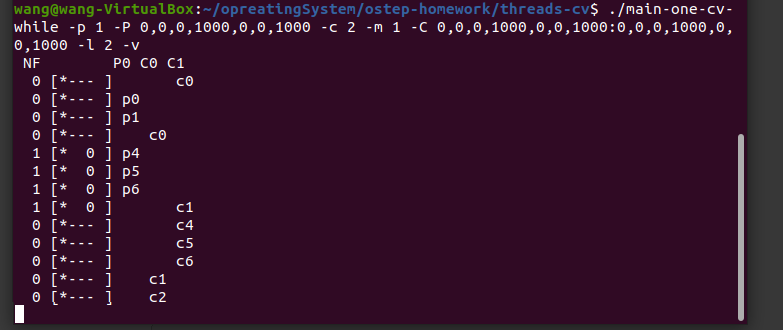


每次运行得到的结果不同，消费者要读取十个数据，所以至少会休眠十次。而程序运行的时间数量级相对于秒来说很小很小，几乎可以忽略不计，所以程序运行的总是长可以看作是休眠的总时长

30.8

只有一个生产者和一个消费者时，生产者产出产品唤醒消费者然后睡眠，消费者消费了产品唤醒生产者再睡眠。周而复始。因此，这种方案没什么问题，因此无法让代码运行出现问题

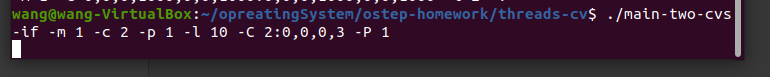
30.9



消费者2发现没有产品直接进入睡眠状态

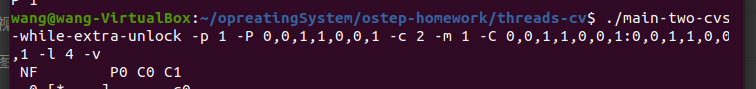
30.10

假设有两个消费者和一个生产者，生产者产生产品之后唤醒消费者1,但是消费者2抢先执行消费了产品，那么消费者1发现缓冲区为空，无法消费，则发生了错误

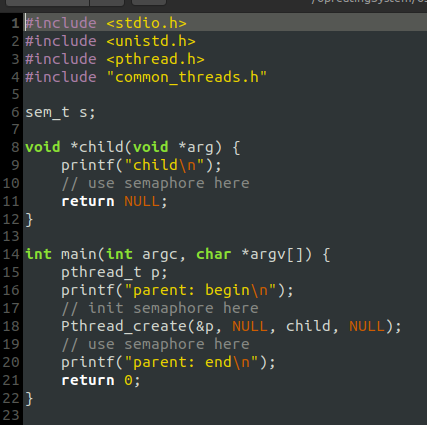


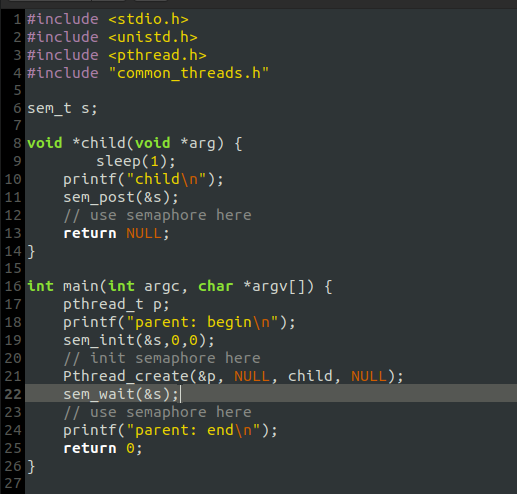
30.11

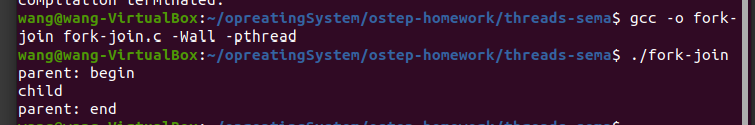
在生产者代码中fill（）没有加锁，在消费者代码中，get（）没有加锁，如果程序发生中断导致消费者顺序被抢先，会发生无元素可取的情况，导致错误的产生



31.1

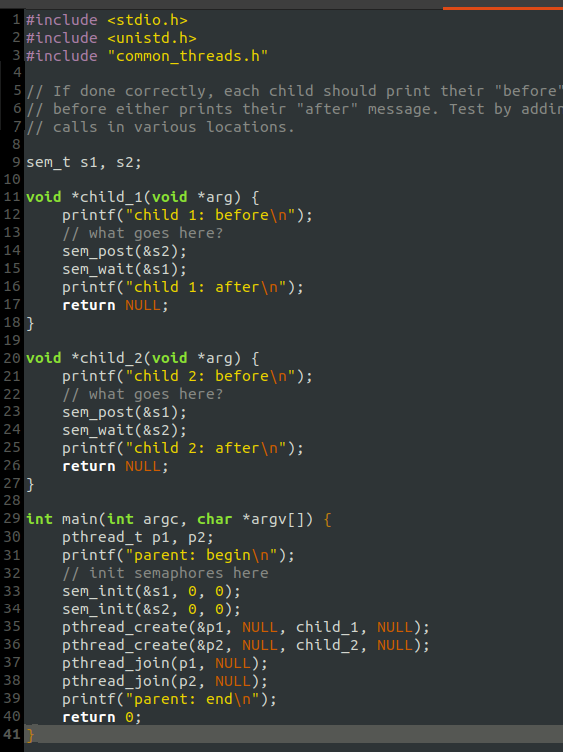


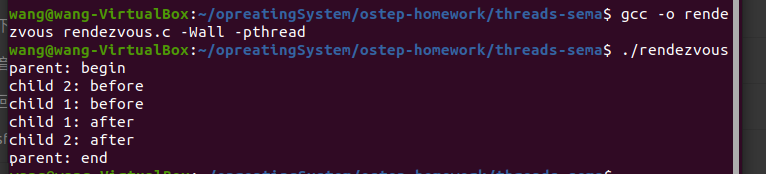




加入了信号量并且在child中停顿1s

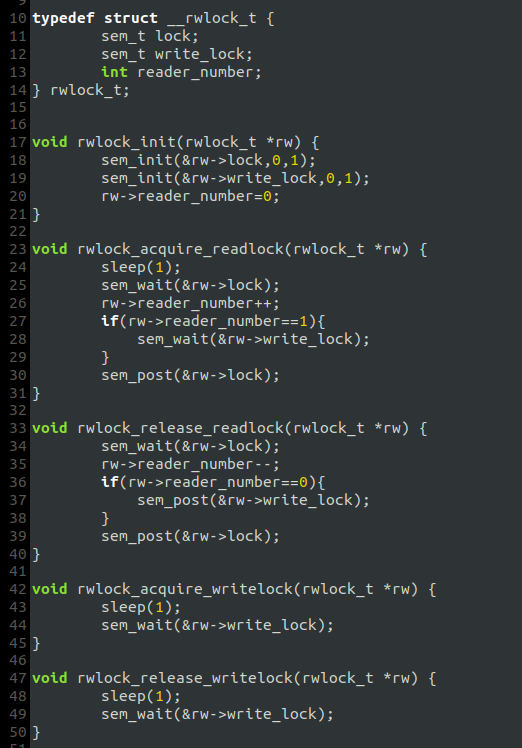
31.2

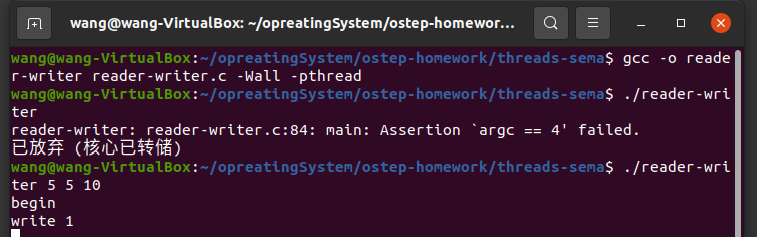




先打印两个before再打印两个after

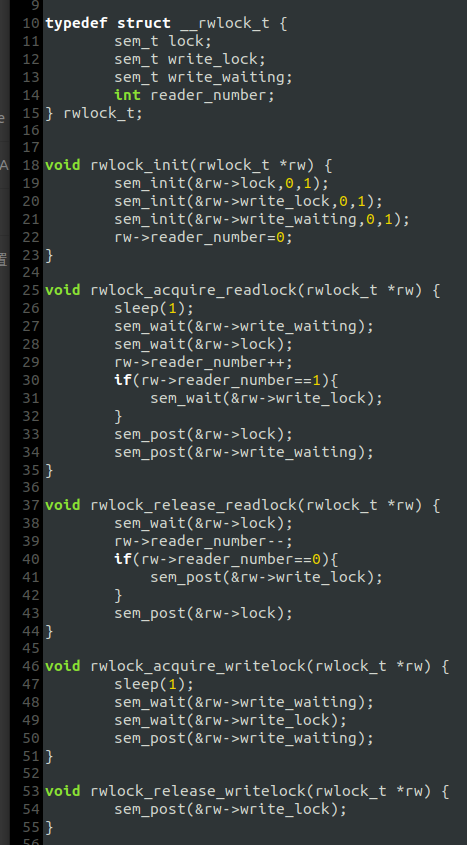
31.4

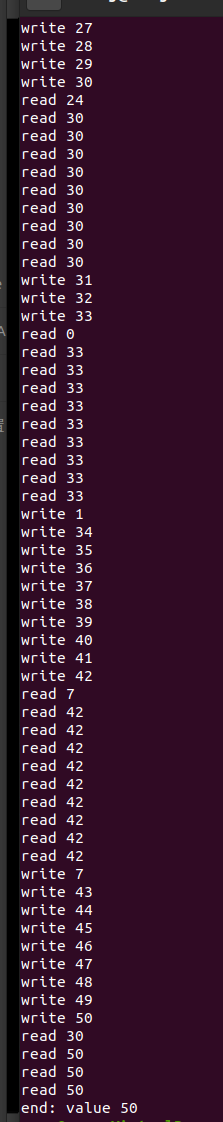




程序三个参数分别为读者数，写者数，每个读者、写者进行的读写操作数  
当读者数量远大于写者时，写者可能饿死，读者不需要锁就能进入临界区，只要有一个读者获得锁，  
其他读者线程就能运行，读者数量可能一直大于0，而写者始终无法获取锁

31.5





新增 write\_waiting 锁，写者读者都需要竞争这个锁。新增读者要持有这把锁，写者进行写也要持有这把锁，一旦写者抢到了这把锁，就不能增加新读者，直到写者完成写

31.6

