**北京邮电大学软件学院、**

**2017-2018学年第1学期实验报告**

**课程名称： 操作系统**

**实验名称：** 实验三 多线程实现生产者**-**消费者问题

**实验完成人：**

**姓名：**\_\_\_\_\_苏可欣\_\_\_**学号：**\_\_\_2016211954\_\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_陈晋鹏\_\_\_\_\_\_\_\_\_\_

**日 期： 2018 年 4 月 30 日**

1. **实验目的**

（说明通过本实验希望达到的目的）

利用PV操作解决生产者消费者问题

1. **实验内容**

（说明本实验的内容）

生产者写入缓冲区和消费者从缓冲区读数的具体流程，生产者首先要获得互斥锁，

并且判断写指针+1 后是否等于读指针，如果相等则进入等待状态，等候条件变量 notfull;如果不等则向缓冲区中写一个整数，并且设置条件变量为 notempty，最后释放互斥锁。消费者线程与生产者线程类似。流程图如下 :

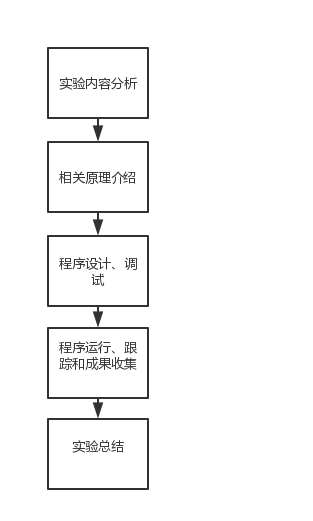
1. **实验环境**

（说明本实验需要的环境）

在Linux下，采用以Vi +GCC+GDB为开发环境的C语言实验环境。其中Vi 作为编辑

器，GCC作为编译器，GDB作为调试器

1. **实验过程描述**



\_个人单独完成\_

实验内容分析

该生产者消费者问题要求用信号量的方式实现。需要一个互斥锁（对原型缓冲区的读写不可同时进行），和两个条件变量来进行协作（即非空和非满的信号）

相关原理介绍

在程序的代码中大量的使用了线程函数，如 pthread\_cond\_signal、

pthread\_mutex\_init、pthread\_mutex\_lock 等。下面简单介绍，详细的说明请查阅资料。

● 线程创建函数:int pthread\_create (pthread\_t \* thread\_id, \_\_const pthread\_attr\_t \*

\_\_attr, void \*(\*\_\_start\_routine) (void \*),void \*\_\_restrict \_\_arg)

● 获得父进程 ID:pthread\_t pthread\_self (void)

● 测试两个线程号是否相同:int pthread\_equal (pthread\_t \_\_thread1, pthread\_t

\_\_thread2)

● 线程退出:void pthread\_exit (void \*\_\_retval)

● 等待指定的线程结束:int pthread\_join (pthread\_t \_\_th, void \*\*\_\_thread\_return)

● 互斥量初始化:pthread\_mutex\_init (pthread\_mutex\_t \*,\_\_const pthread\_mutexattr\_t \*)

● 销毁互斥量:int pthread\_mutex\_destroy (pthread\_mutex\_t \*\_\_mutex)

● 锁定互斥量(阻塞)::int pthread\_mutex\_lock (pthread\_mutex\_t \*\_\_mutex)

● 解锁互斥量:int pthread\_mutex\_unlock (pthread\_mutex\_t \*\_\_mutex)

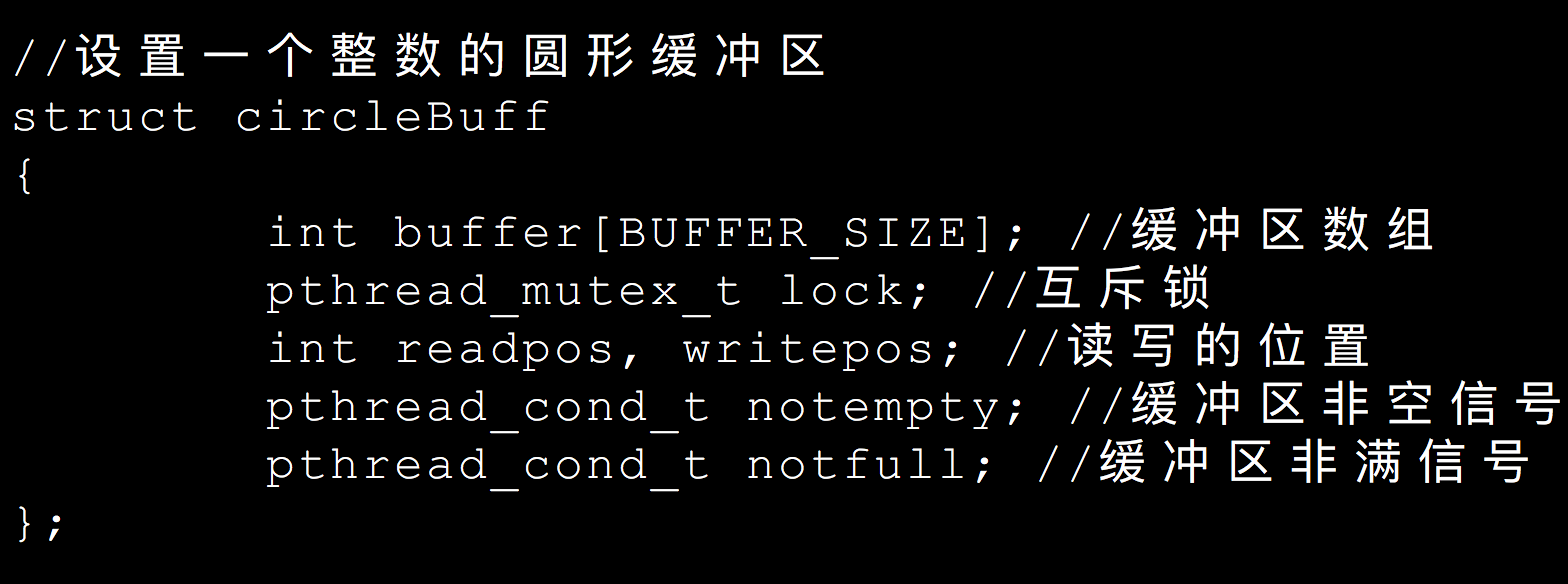
● 唤醒线程等待条件变量:int pthread\_cond\_signal (pthread\_cond\_t \*\_\_cond)

● 等待条件变量(阻塞)::int pthread\_cond\_wait (pthread\_cond\_t \*\_\_restrict \_\_cond,

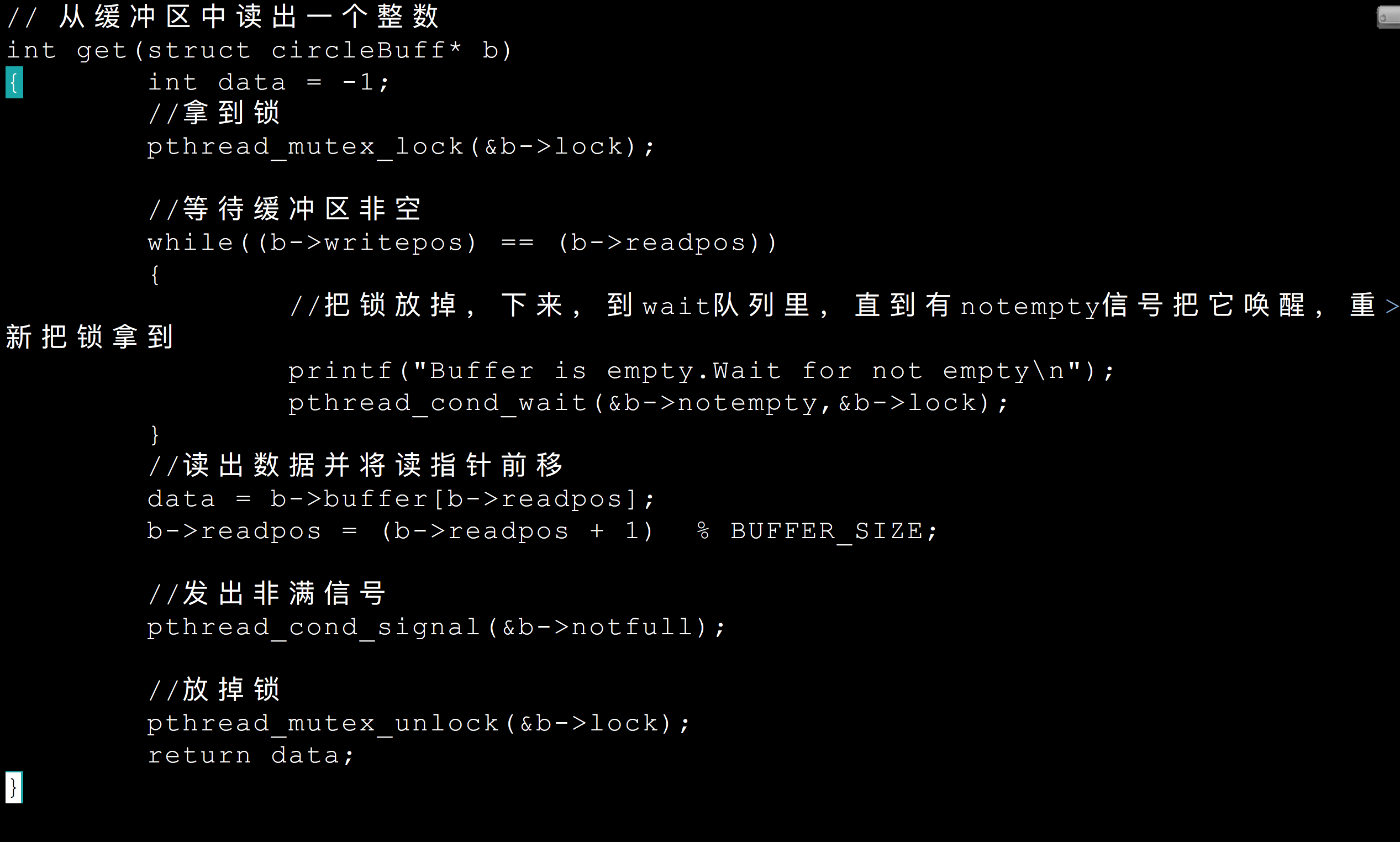
总体来说就是老师上课讲的实现互斥的方法中的信号量一栏，并且h爱是记录型信号量，即不会陷入忙等，而是会由block到waiting queue，再唤醒的。

程序设计

* 资源的数据结构



* 消费者消费



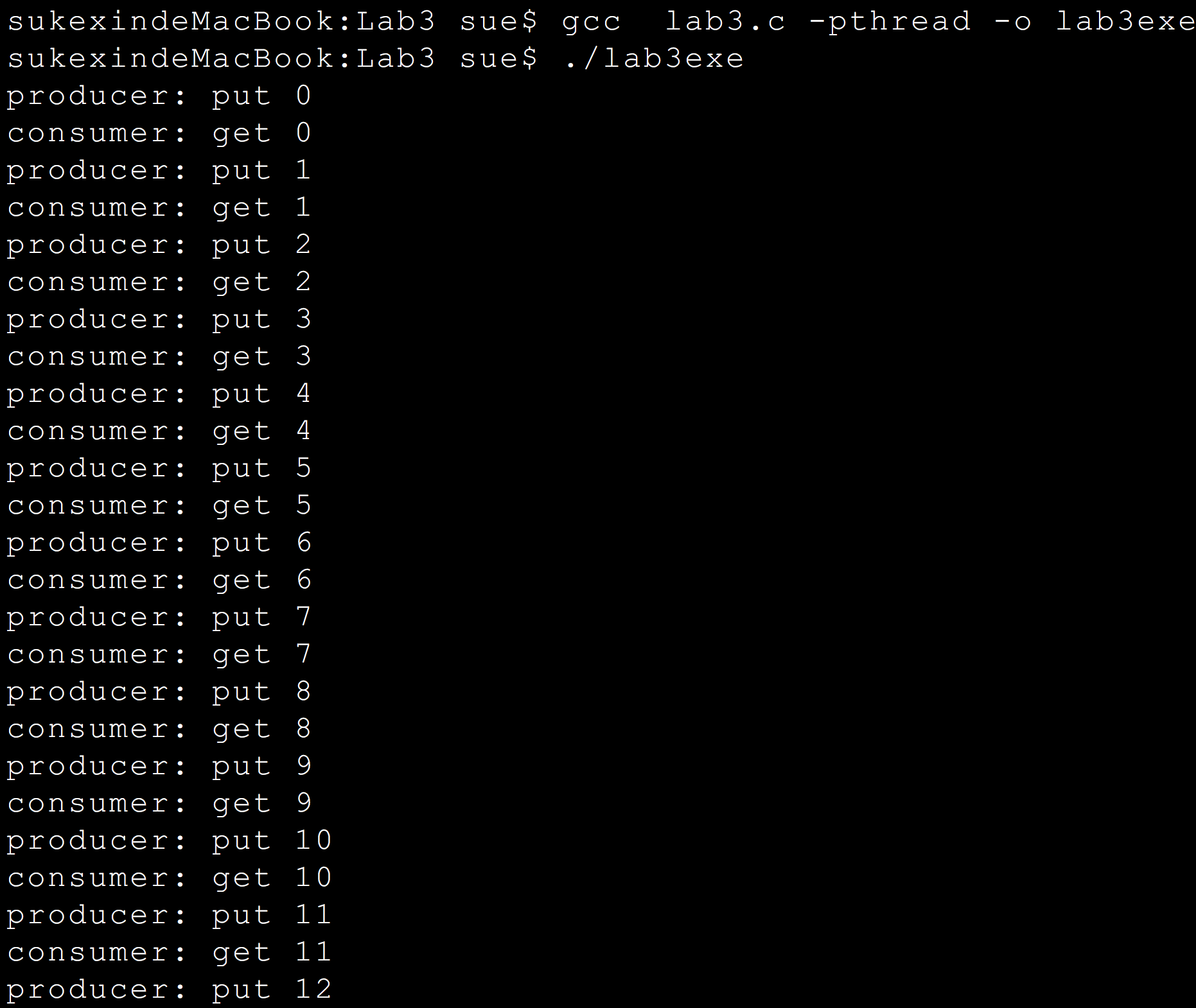
* 生产者生产

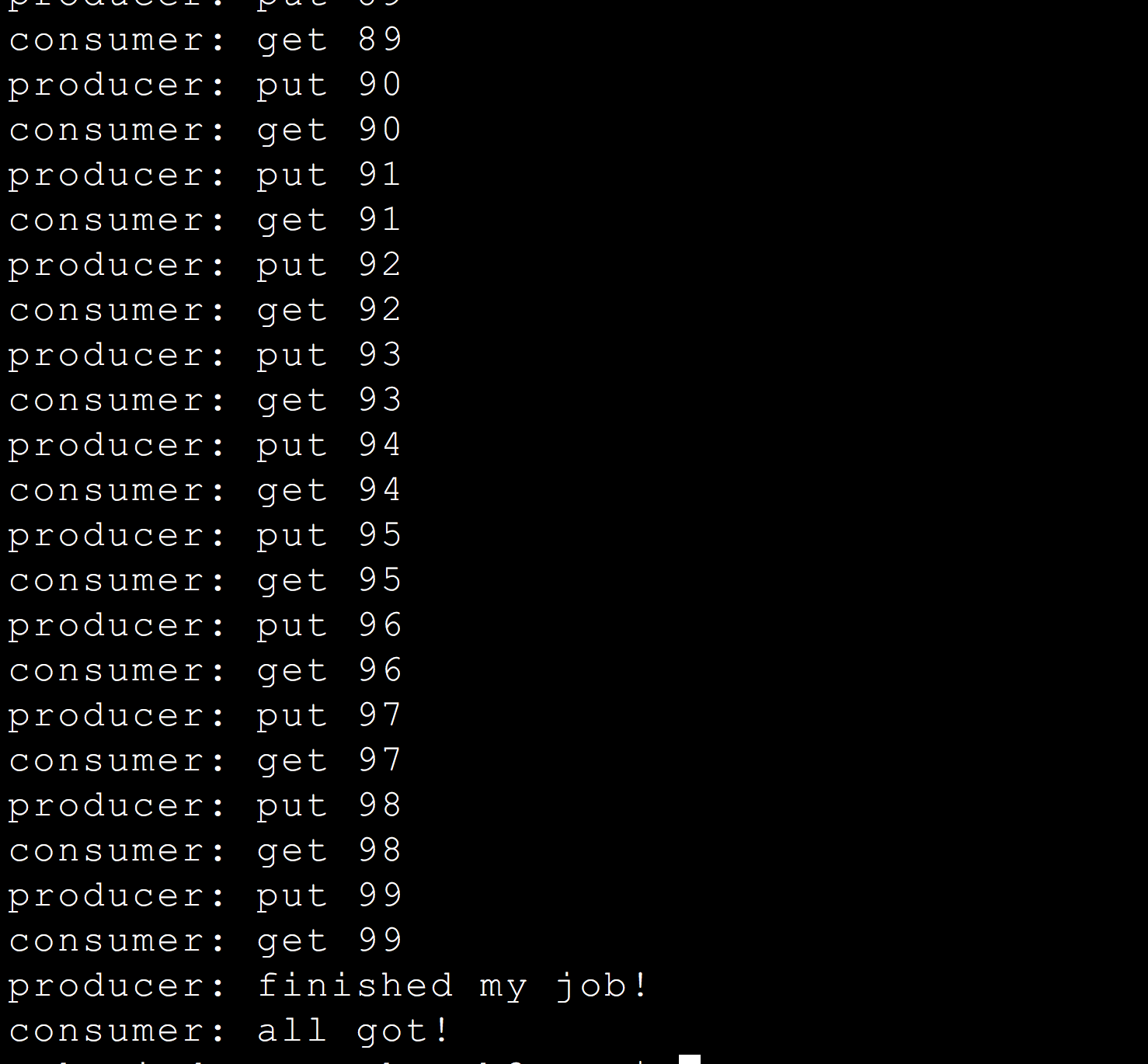


调试

* 在调试过程中遇到一些问题，首先是编译的时候pthread找不到的问题。通过在编译的时候加上 -pthread ，成功链接和添加宏。
* 还有就是我自己的问题，导致奇怪的bug：生产者正常生产（1-100编号），消费者一直取的是0。有点迷，经过断点，单步调试，发现时b->buffer[b->readpos] 写成了b->readpos，b->buffer[b->writepos]写成了b->writepos

程序运行结果

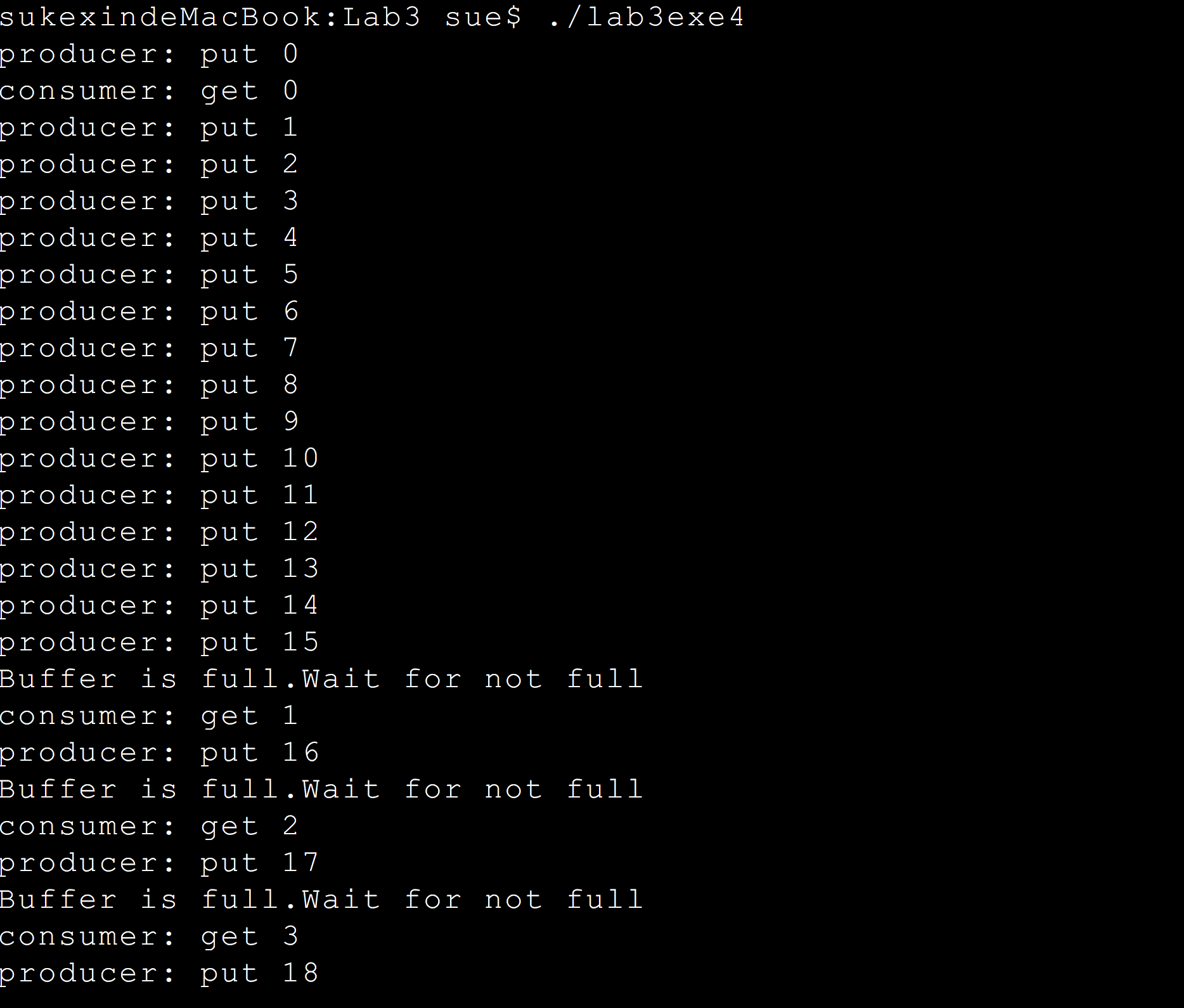




然而这让我懵了 ！！！！因为 这个结果是在太奇怪了！从头到尾，producer和consumer都是生产一个，消费一个，生产一个，消费一个，完全同步，让我觉得是我哪里写错了，导致有synchronize的地方，这样的效率并不高。

为了来测试我是不是搞错了，我在consumer里让他每获取一个就sleep个5ms，这样的话，要是不是synchronize必然会有producer生产太快，导致缓冲区满了的情况

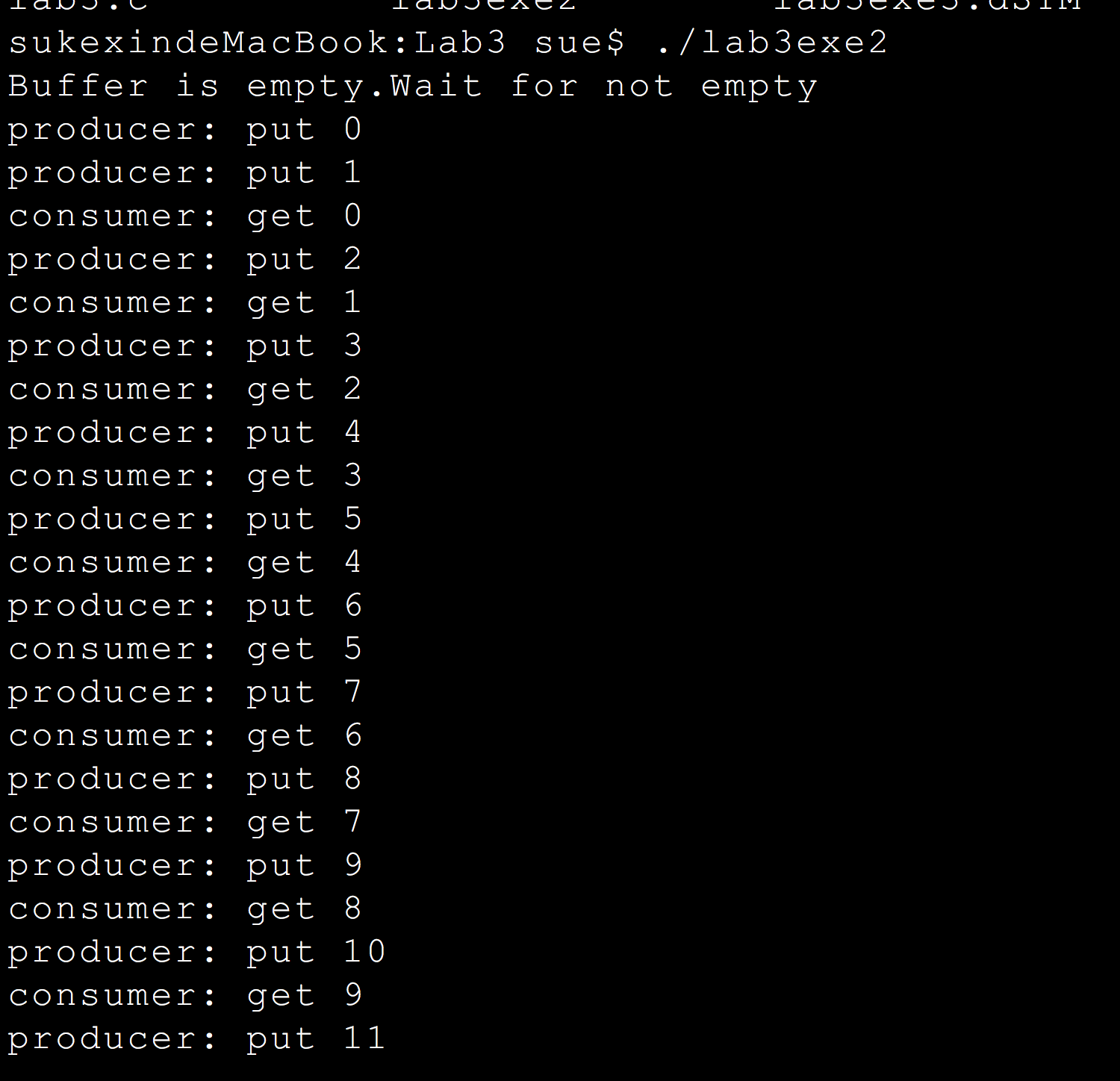
于是测试 结果如下图

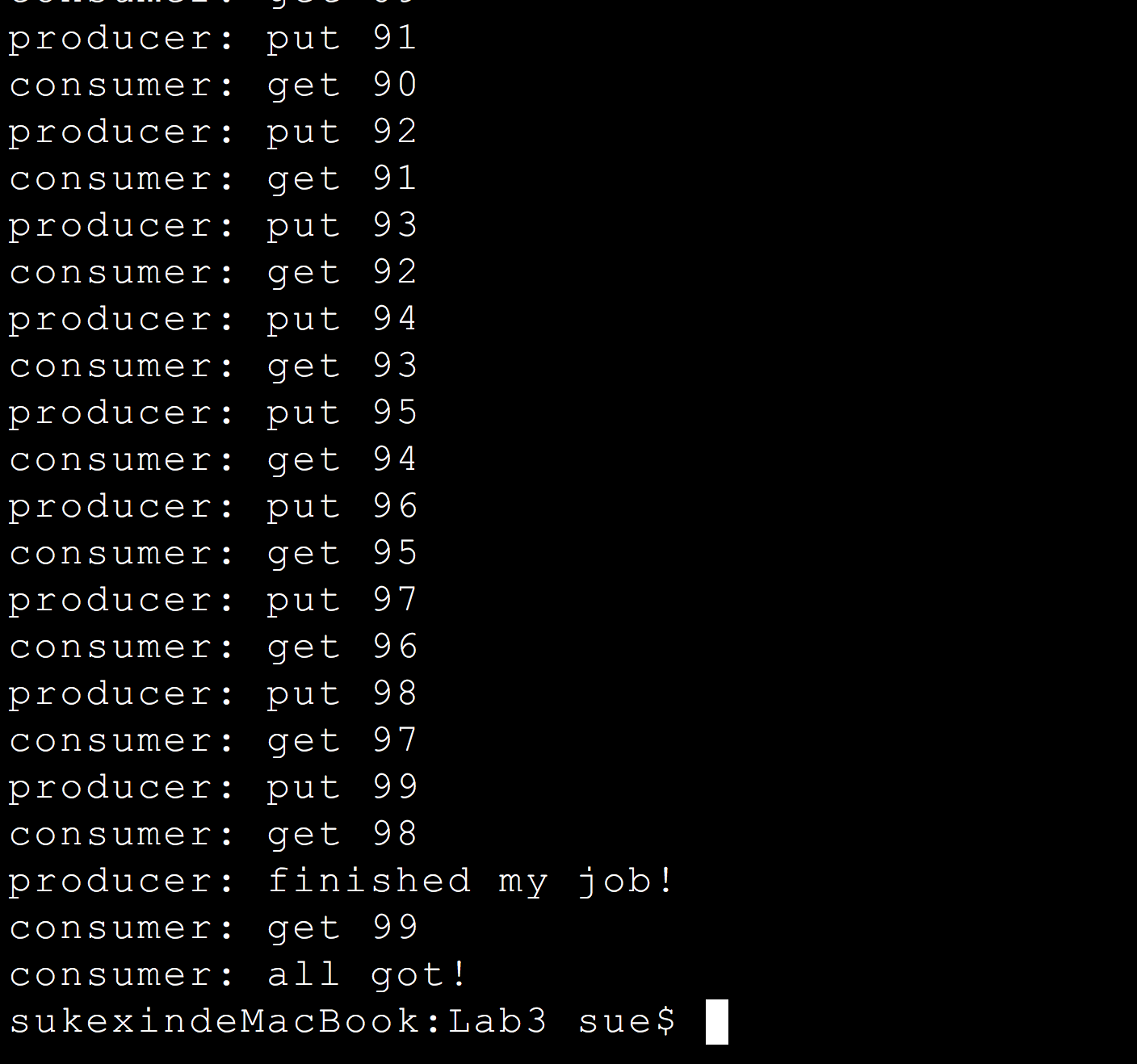


这证明了原来的代码没有问题， 只是真的恰好同步，时间片都差不多。

删掉原来的sleep(5)

再运行一次





正常了！

1. **实验结果**

（说明实验完成情况）

满足题目要求，代码易读整洁，注释清晰

运行截图见标题3

1. **附件**

**6.1 附件1：源代码**

**6.2 附件2：XXX**