《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李欣宇 | 学号 | U201911658 | 专业班级 | 信安1901 | 时间 | 2021.12.26 |

**一、实验目的**

1）理解进程/线程的概念和应用编程过程；

2）理解进程/线程的同步机制和应用编程；

**二、实验内容**

1）在Linux下创建一对父子进程。

2）在Linux下创建2个线程A和B，循环输出数据或字符串。

3）在Windows下创建线程A和B，循环输出数据或字符串。

4）在Linux下创建一对父子进程，实验wait同步函数。

5）在Windows下利用线程实现并发画圆/画方。

6）在Windows或Linux下利用线程实现“生产者-消费者”同步控制

7）在Linux下利用信号机制实现进程通信。

8）在Windows或Linux下模拟哲学家就餐，提供死锁和非死锁解法

1,6,8必做,其余任选1。本次实验选了4

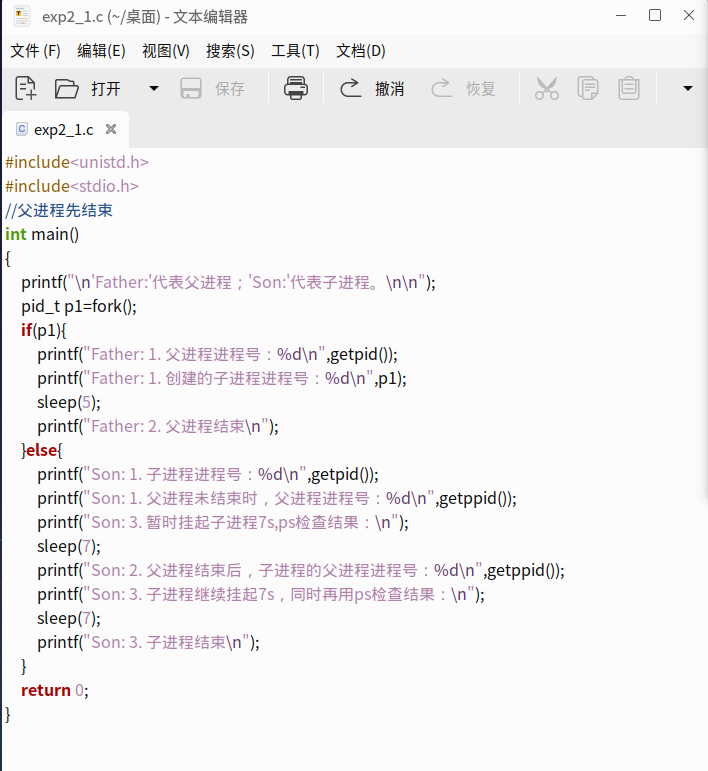
**三、实验过程**

**3.1 Linux下创建一对父子进程**

（1）源码

使用fork函数创建子进程，父进程输出标识符为“Father”,子进程输出标识符为“Son”，fork的返回值是子进程的pid号，对于子进程中得到的pid号是0，根据这个可以将子进程和父进程进行的操作分开，在子进程中，使用getppid在获取父进程pid号，使用getpid获取自身进程号。

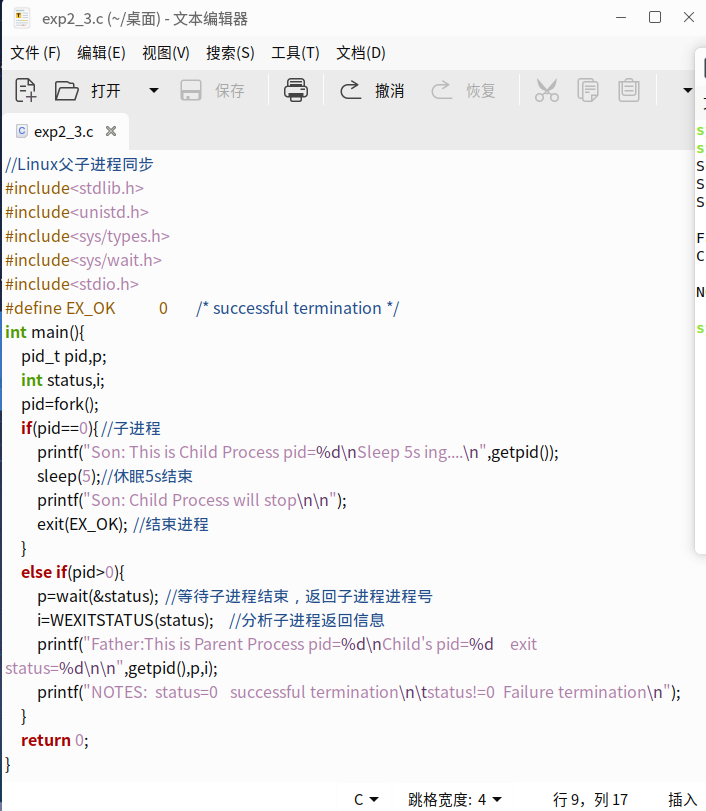
该程序让父进程先结束，然后子进程再结束，其中使用sleep进行进程休眠，父进程休眠时间比子进程休眠时间短，所以会先于子进程结束，采用不用的sleep长度，便于ps进行查看。



**3.2 Linux下使用wait同步父子进程**

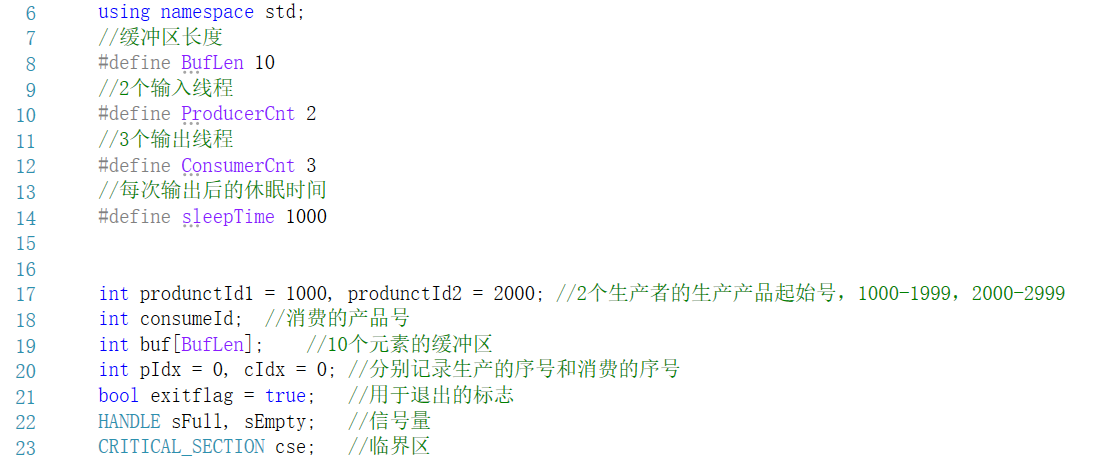
（1）源码

使用 fork()函数创建子进程，利用 fork 返回值来判断当前进程是父进程还是子进程。其中，子进程休眠 5 秒之后用 exit()函数返回参数；而父进程不休眠，使用 wait()函数等待子进程先结束，并将子进程的进程号以返回值的形式记录到变量 p 中，将子进程的返回参数记录到变量 status 中，并通过 WEXITSTATUS()函数进程分析，最终由父进程输出子进程的返回信息。具体代码如下：



**3.3 Windows实现“生产者-消费者”同步**

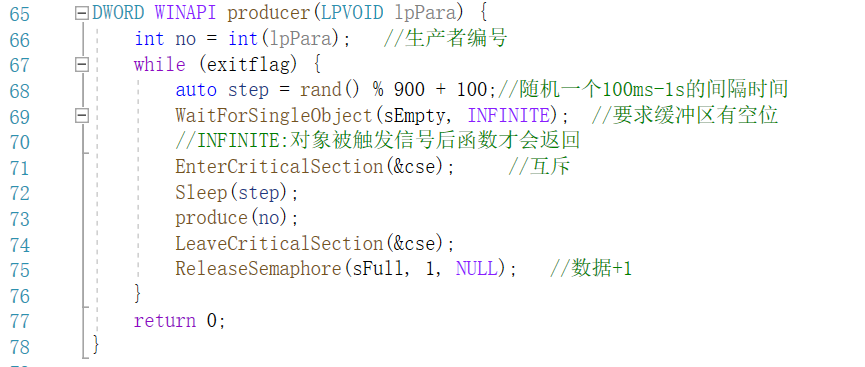
Windows 环境下，首先定义全局变量，包括缓冲区长度、生产者消费者个数、生产产品号消费产品号等，该部分代码如下：



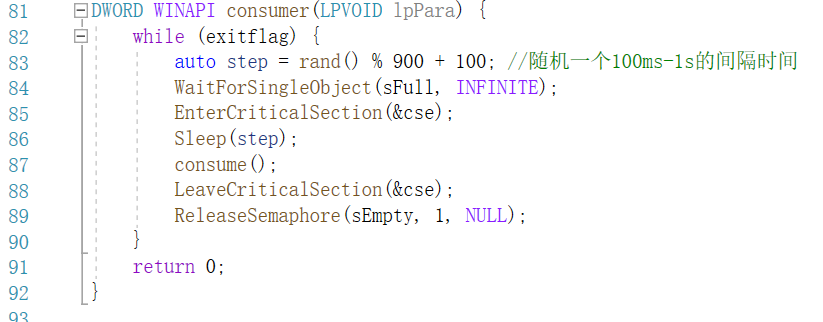
main 函数里使用 CreateSemaphore()函数创建信号量 sFull 和 SEmpty 分别表示缓冲区数据个数和缓冲区空位个数，并设置信号量的初始值和最大值，使用InitalizeCriticalSection() 函数初始化临界区变量 cse 用于缓冲区互斥使用。然后使用 CreateThead()函数分别创建消费者线程和生产者线程。最后 main 函数主线程进入死循环，直到有输入时停止。



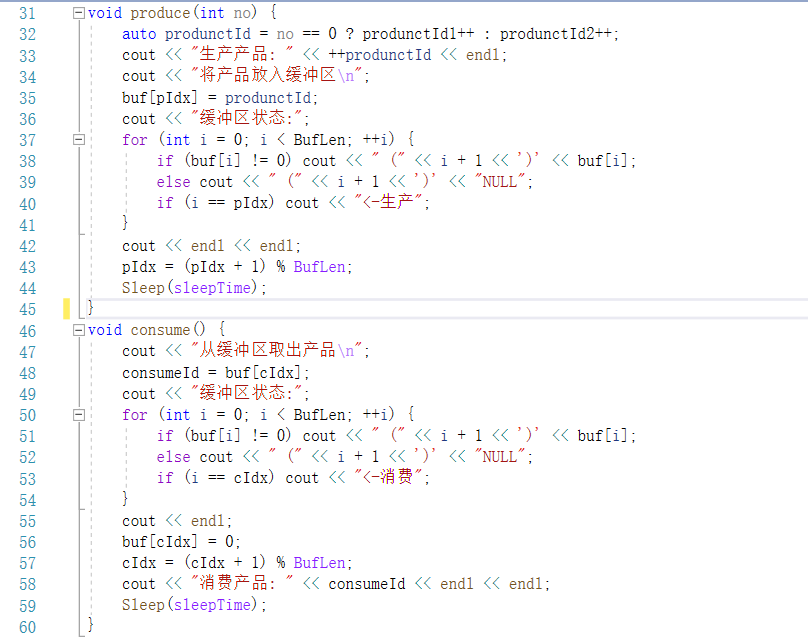
生产者线程为函数 producer()，使用 lpPara 参数传递生产者的序号，1 号生产者生产1000~1999号产品，2号生产者生产2000-2999号产品。每次生产产品的过程如下：生产产品前使用WaitForSingleObject()函数作为P操作等待缓冲区有空位的信号量sEmpty，使用EnterCriticalSection()函数进入临界区，对缓冲区进行互斥使用。使用produce()函数生产产品。生产产品后，使用函数LeaveCriticalSection()离开临界区。最后使用ReleaseSemaphore()函数作为V操作使缓冲区个数+1



消费者线程为函数consumer()。每次消费产品过程如下：消费产品前使用WaitForSingleObject() 函数作为 P操作等待缓冲区有产品的信号量sFull，使用EnterCriticalSection()函数进入临界区，对缓冲区进行互斥使用。使用consume()函数生产产品。生产产品后，使用函数LeaveCriticalSection()离开临界区。最后使用ReleaseSemaphore()函数作为V操作使缓冲区空位+1。



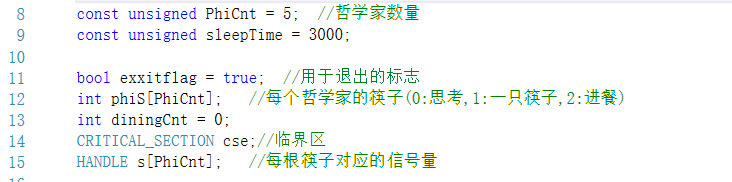
produce()和consume()分别用于生产和消费产品，并在控制台上输出相应产品在整个缓冲区的状态。其中，缓冲区采用的是循环队列，生产和消费产品都是从缓冲区的 0 号~9 号再循环回 0 号。



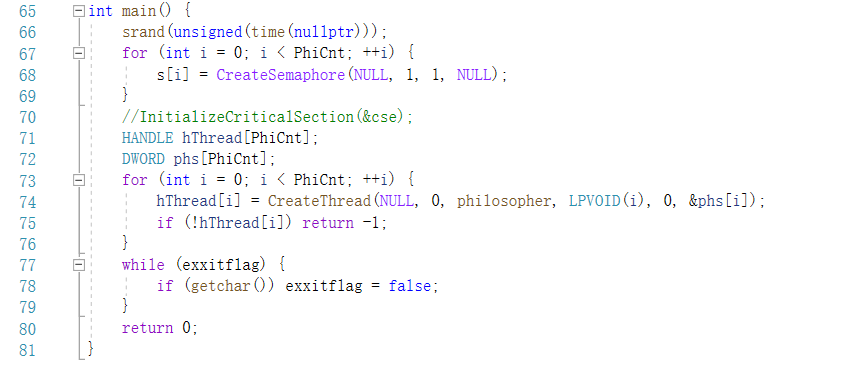
**3.4 Windows 下模拟哲学家就餐**

（1）死锁解法

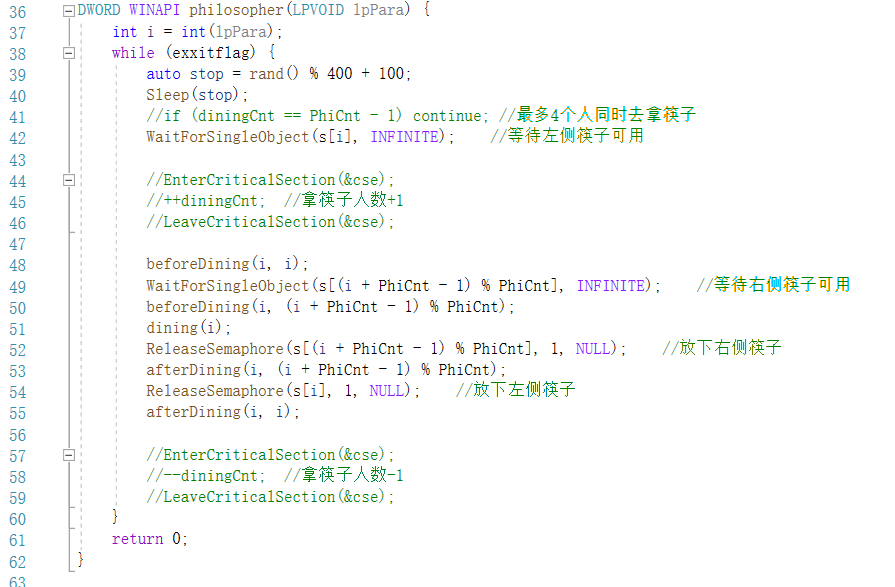
在Windows下，首先定义全局变量，包括哲学家数量，每个哲学家的筷子。



主函数中，使用CreateSemaphore()函数为每个筷子初始化信号量s[i]，表示每根筷子是否可用（1可用，0不可用），然后创建每个哲学家的线程。然后进入死循环，直到有exxitflag为false时停止。



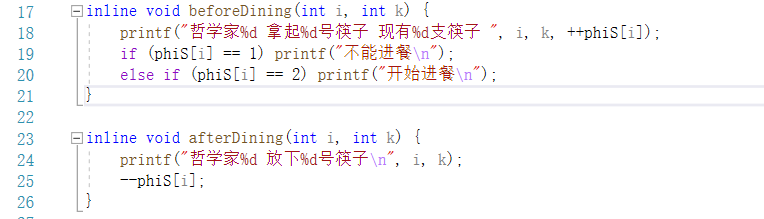
哲学家线程为函数philosopher()，进行循环思考和进餐。每次就餐过程，先要使用WaitForSingleObject()函数作为P操作，依次等待左侧筷子和右侧筷子的信号量可用，然后使用函数dining()进餐，然后ReleaseSemaphore()，依次放下右侧筷子和左侧筷子，作为V操作使两者对应信号量可用。

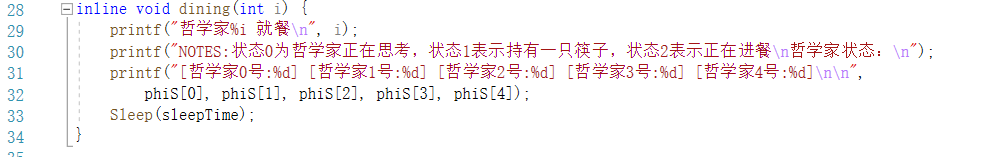


beforeDining()中，哲学家拿筷子并输出信息，

afterDining()哲学家放筷子并输出信息

dining()表示哲学家在进餐，并输出全部哲学家当前筷子数（状态）。





以上这种就餐情况可知 ，若0-4号哲学家都拿起了左侧的筷子，则会造成“死锁”现象，即每个哲学家都在等待右侧的筷子被放下，而一直无法进餐。

（2）非死锁解法

死锁的避免可以从产生死锁的四个必要条件进行分析，这里采用了避免环路条件产生，即最多允许4个哲学家取筷子，这样可以保证至少一个哲学家能够顺利进餐，从而不会产生死锁。

这里使用diningCnt变量作为计数变量，最大为4，同时这个变量是一个临界资源，所以要使用一个临界区cse，以确保不同线程对diningCnt的访问时互斥的，程序整体与死锁解法一致，仅在philosopher（）函数中加上diningCnt和cse部分，具体代码如下。



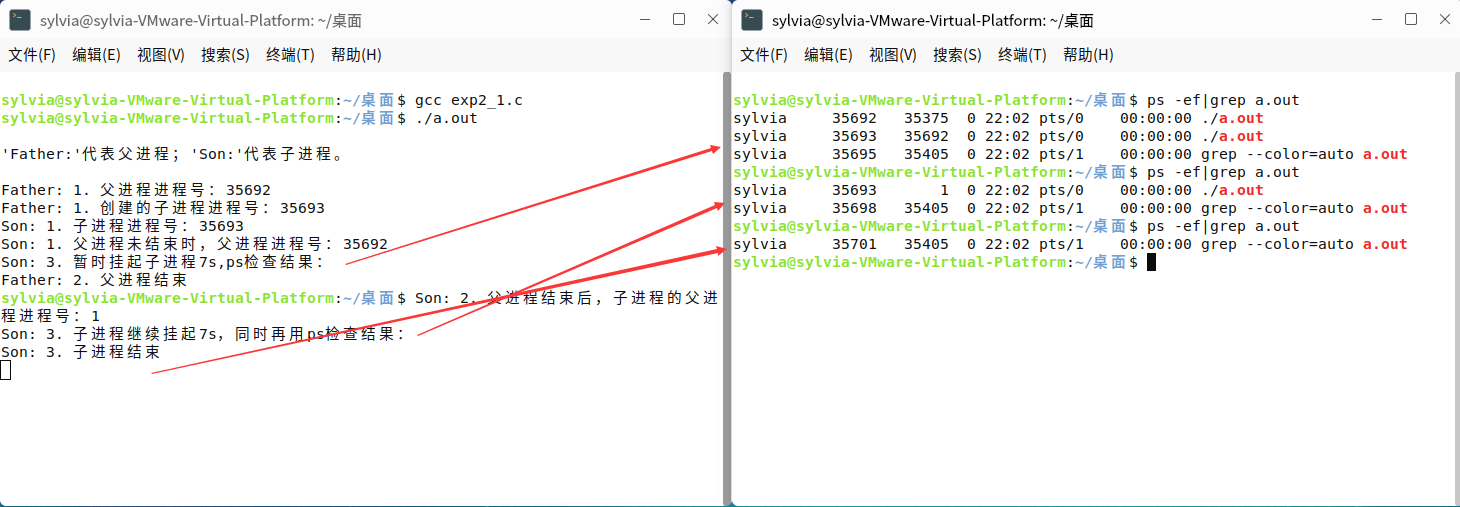
**四、实验结果**

**4.1Linux下创建一对父子进程**

创建的父进程pid是35692，子进程pid号是35693，父进程未结束前可以看到子进程的父进程号是35692.

父进程结束后，子进程的pid号变为1，也就是init进程

子进程也结束后，可以看到已经没有35693进程了

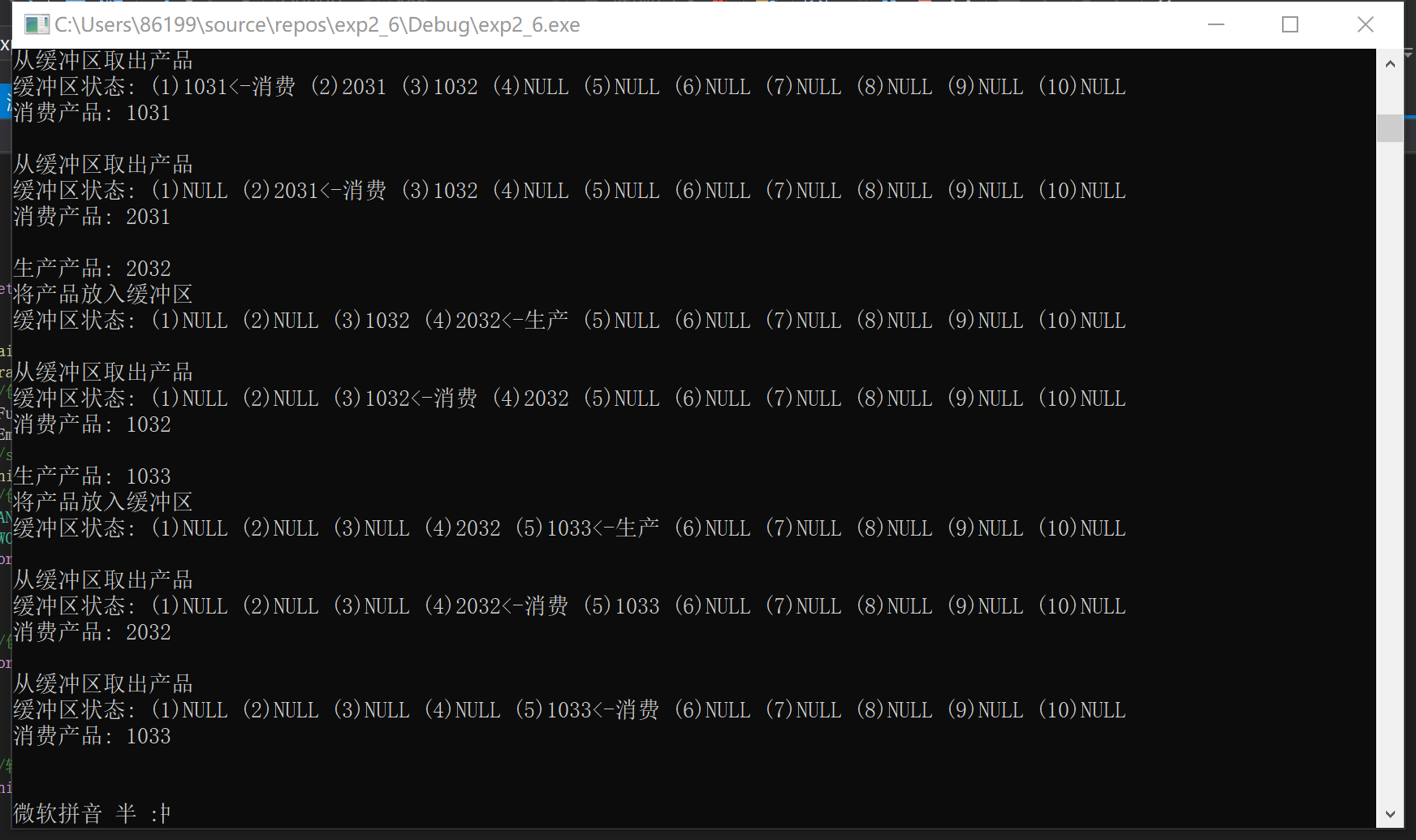
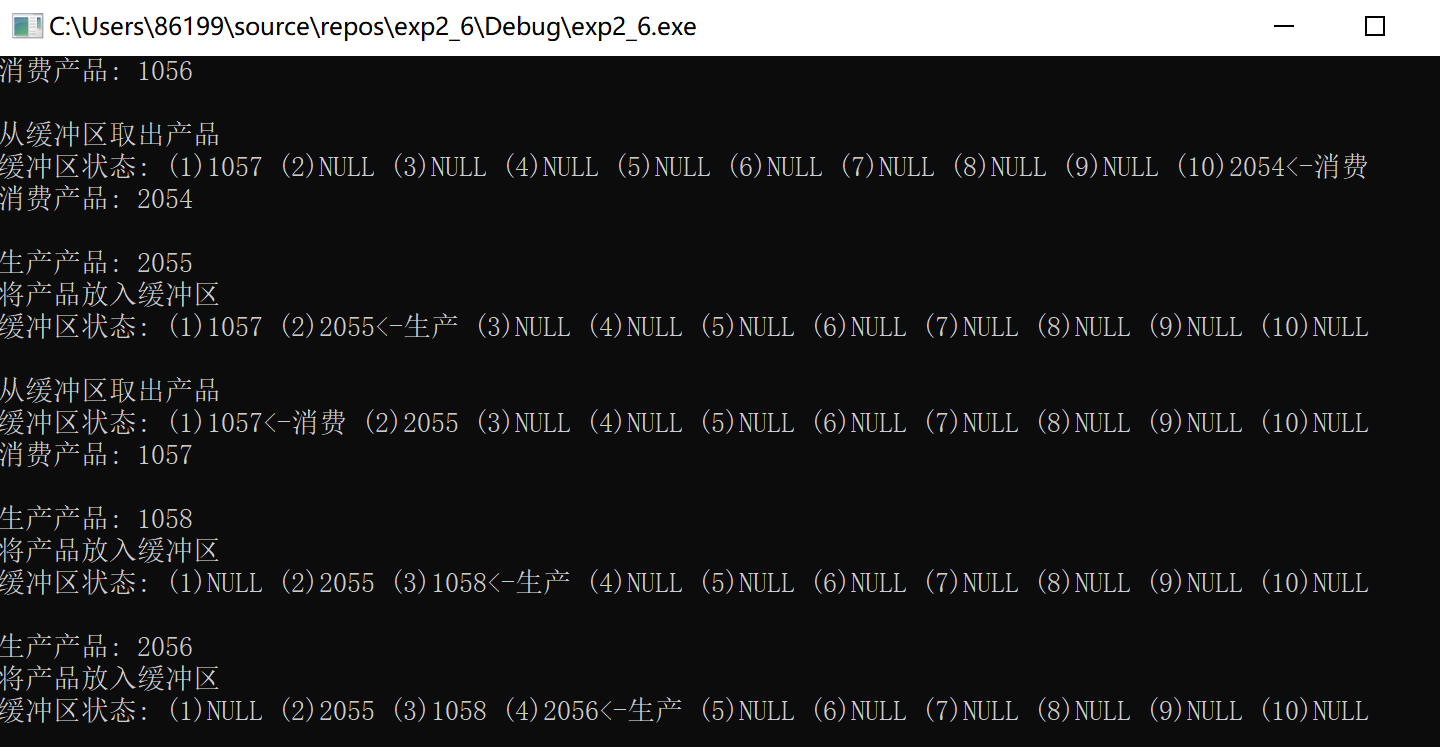
****

**4.2 Linux下使用wait同步父子进程**

使用wait进行同步，子进程结束后父进程在wait中接收到信息，状态由挂起转为就绪（运行），从status 获取已经结束的子进程的一些信息，包括进程号和结束状态。

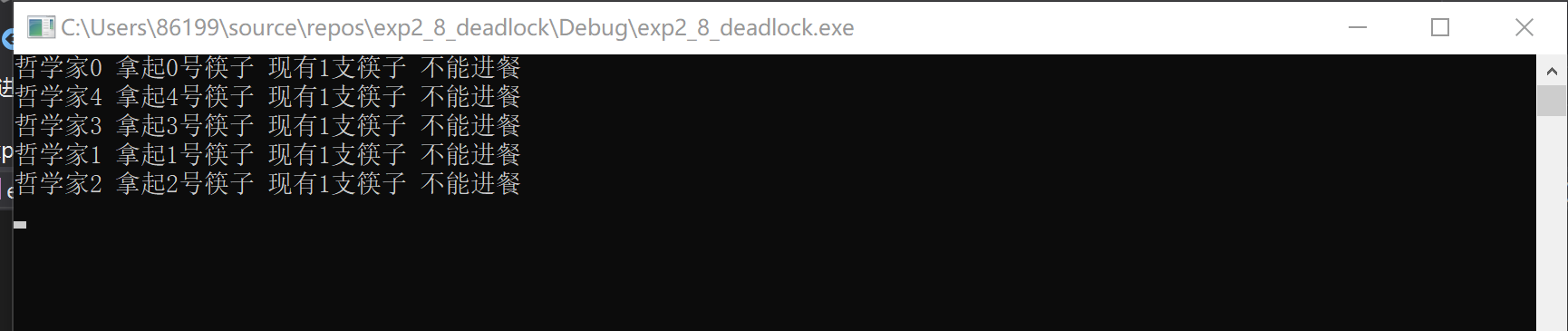


**4.3 Windows实现“生产者-消费者”同步**

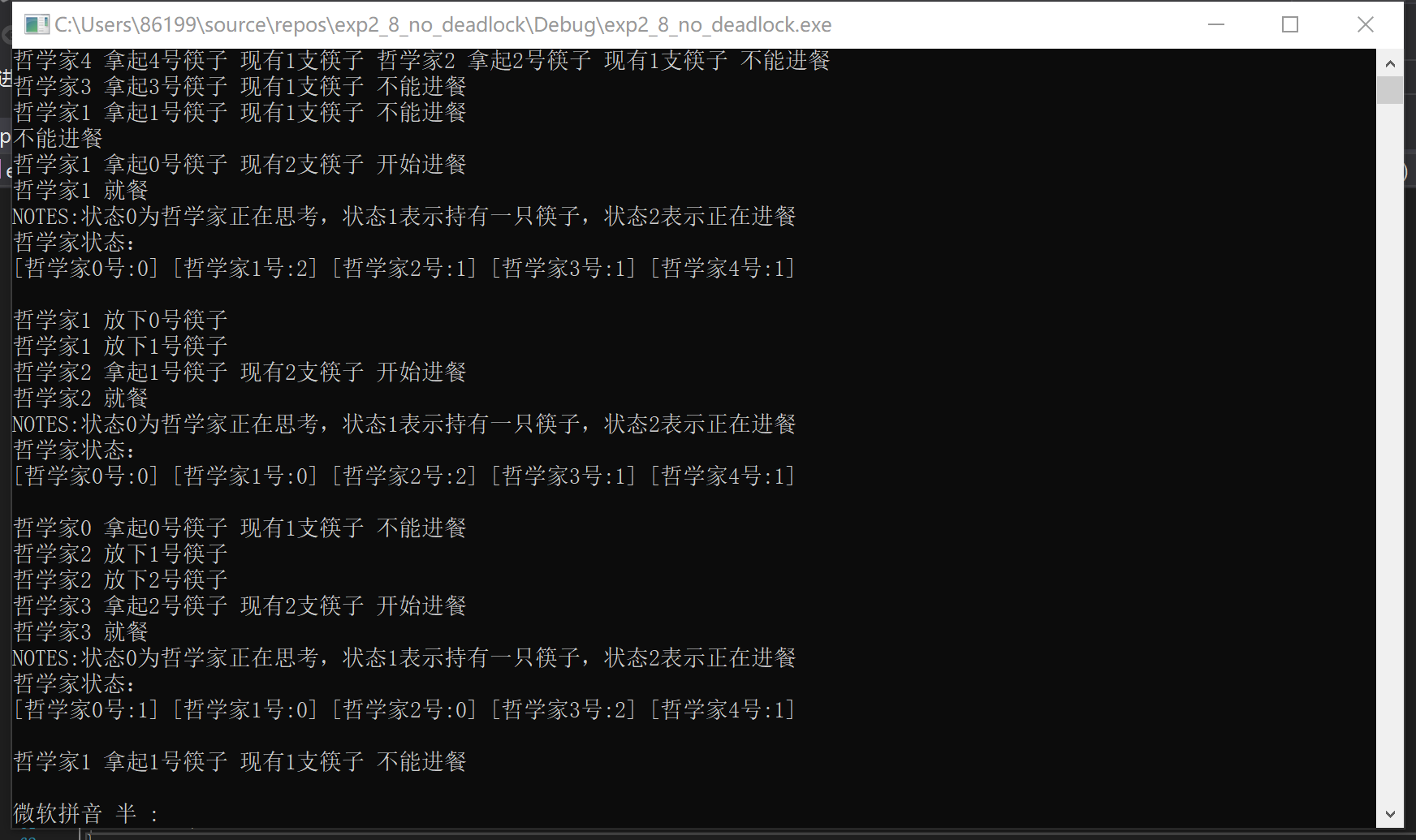
****

**4.4 Windows 下模拟哲学家就餐**

（1）死锁解法

****

（2）非死锁解法

****

**五、实验错误排查和解决方法**

**5.1 Linux下创建一对父子进程**

无

**5.2 Linux下使用wait同步父子进程**

无

**5.3 Windows实现“生产者-消费者”同步**

EnterCriticalSection ,LeaveCriticalSection ,WaitForSingleObject ,ReleaseSemaphore 信号量、线程同步问题在软件安全课设中已经使用过，所以这一部分操作起来比较简单，没有问题

**5.4 Windows 下模拟哲学家就餐**

无

**六、实验参考资料和网址**

（1）教学课件

（2）https://blog.csdn.net/drdairen/article/details/51896141

（3）https://blog.csdn.net/a2796749/article/details/79383088

（4）https://blog.csdn.net/kxjrzyk/article/details/81603049

（5）https://blog.csdn.net/m\_buddy/article/details/73433181

（6）https://blog.csdn.net/s\_lisheng/article/details/74278765

（7）https://www.cnblogs.com/coolcpp/p/thread-synchronous.html

（8）https://blog.csdn.net/xyisv/article/details/80467668

（9）https://blog.csdn.net/weixin\_36440319/article/details/117163701

（10）https://blog.csdn.net/humblehunger/article/details/106593577