《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邬雪菲 | 学号 | U202112131 | 专业班级 | 网安2104 | 时间 | 2023.11.28 |

**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；

3）掌握和推广国产操作系统（推荐银河麒麟或优麒麟，建议）

**二、实验内容**

1）在Linux/Windows下创建2个线程A和B，循环输出数据或字符串。

2）在Liunx下创建（fork）一个子进程，实验wait/exit函数

3）在Windows/Linux下，利用线程实现并发画圆画方。

4）在Windows或Linux下利用线程实现“生产者-消费者”同步控制

5）在Linux下利用信号机制(signal)实现进程通信

6）在Windows或Linux下模拟哲学家就餐，提供死锁和非死锁解法。

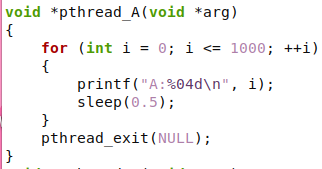
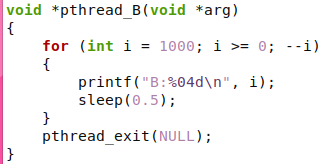
7）研读Linux内核并用printk调试进程创建和调度策略的相关信息。

**三、实验过程**

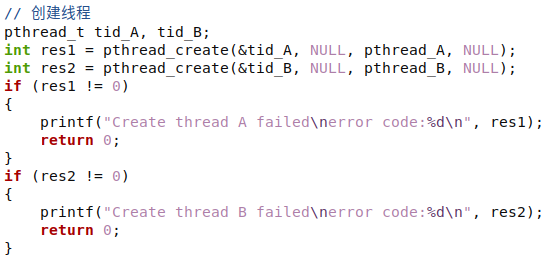
**3.1 双线程循环输出数据**

1）定义线程函数

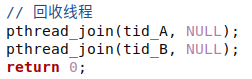
分别定义线程A和线程B需要执行的输出数据函数，一个升序输出一个降序

2）创建线程

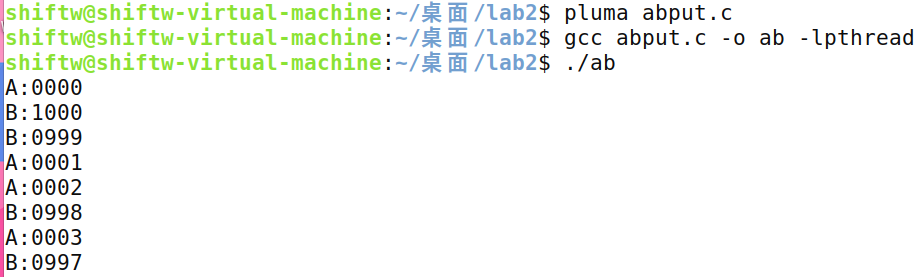


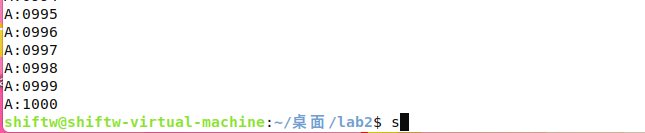
3）回收线程



1. 测试程序

编译运行c程序，发现A和B线程循环输出，但是两者并非严格交替输出，当执行一段时间之后，可明显观察到某一个进程进度快于另一个进程，最终一个进程先行完成所有输出，只留下一个进程单独执行输出数据的操作。

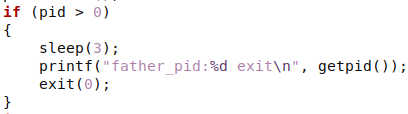




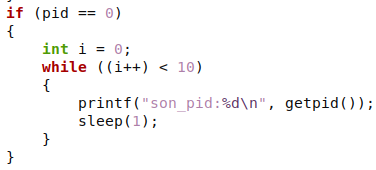
**3.2 Linux实验wait/exit函数**

1）父进程不用wait函数

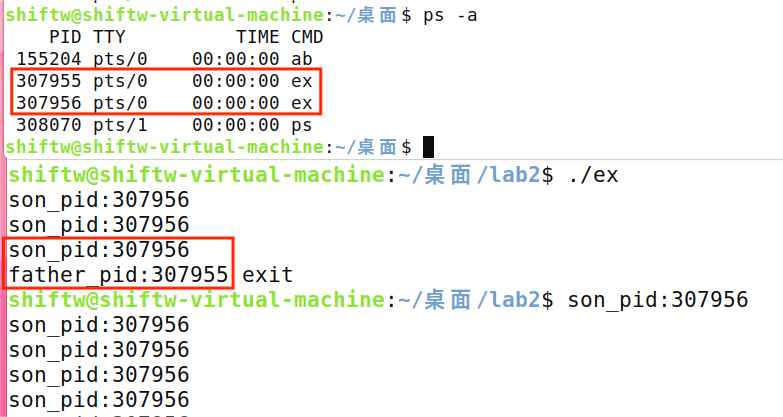
fork()创建子进程后，父进程休眠3秒后使用exit()结束进程并打印进程ID



子进程则进入死循环，使得父进程先于子进程结束

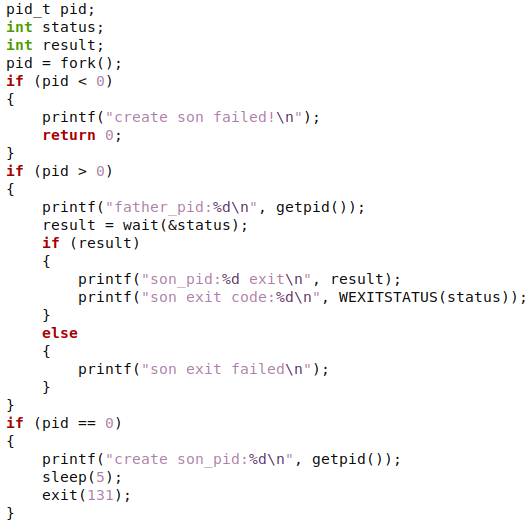


编译运行程序，同时使用ps命令显示进程，观察到显示运行的进程与输出的子进程和父进程ID一致

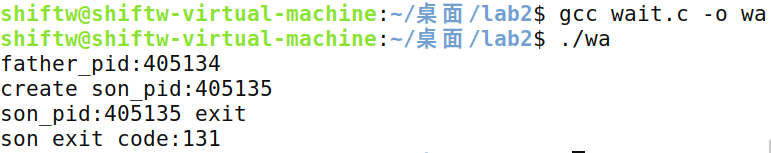


2）父进程用wait函数

创建子进程之后，父进程不使用sleep休眠，而是用wait函数等待回收子进程。子进程休眠5秒后结束，exit()设置特定的返回参数。父进程中接收参数并printf子进程返回的参数。



编译运行程序，可见五秒后子进程结束，父进程正确获取了子进程的返回参数并输出

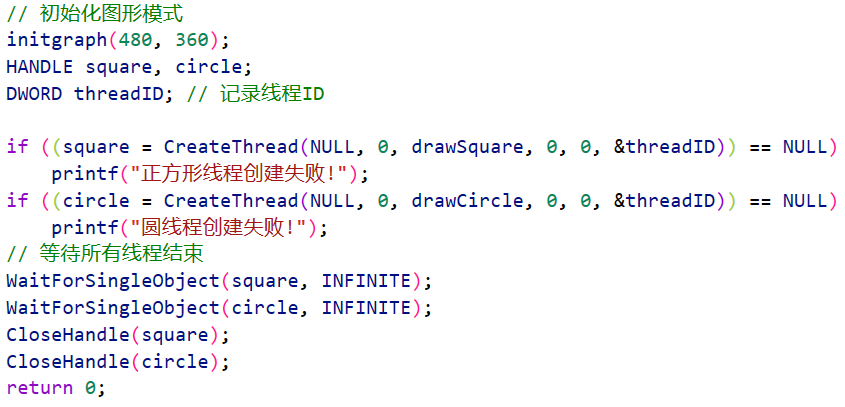


**3.3 Windows并发线程画圆画方**

1）程序编写

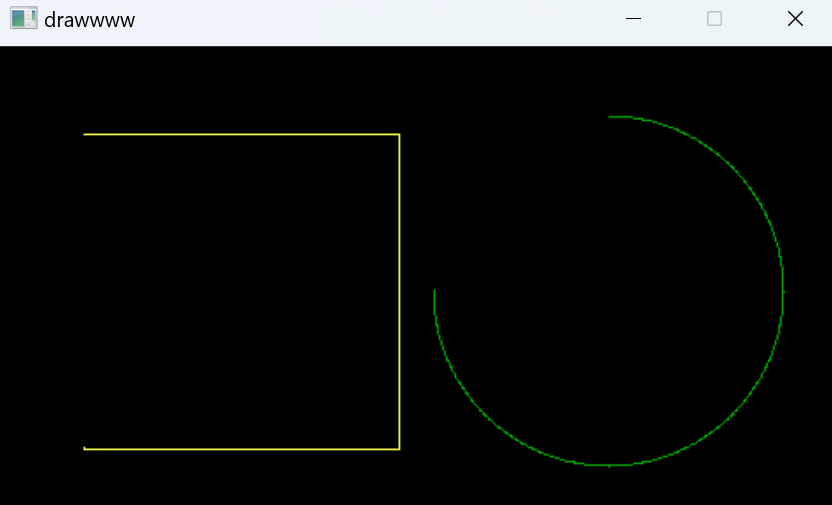
在Windows11的Visual Studio2022平台上进行开发。

首先需要配置画图环境。画图需要使用库graphics.h中的putpixel(x,y,color)等函数来绘制图像，网站下载EasyX头文件安装程序，运行后选择对应的开发环境安装，便可使用Visual Studio创建控制台项目完成程序的编写。EasyX文档中有详细的绘图函数使用教程。



2）运行测试

生成解决方案并运行，可观察到窗口中画圆和画方同时顺时针进行，并且绘制进度同步



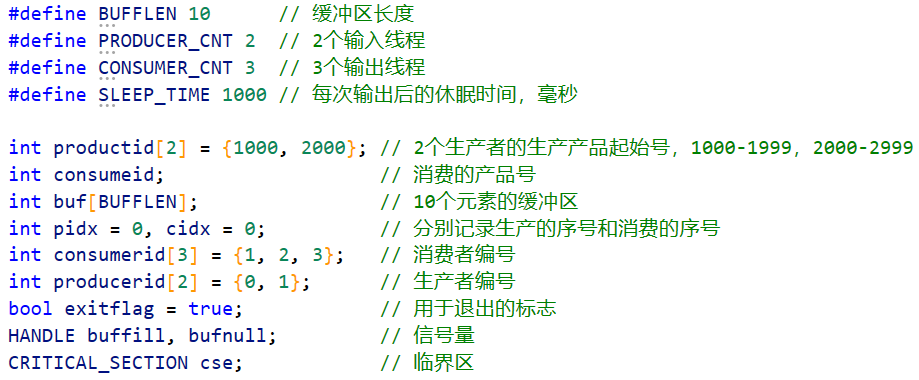
**3.4 Windows“生产者-消费者”同步控制**

1）开发环境

在Windows11下的Visual Studio2022平台上开发控制台项目。

2）主程序编写

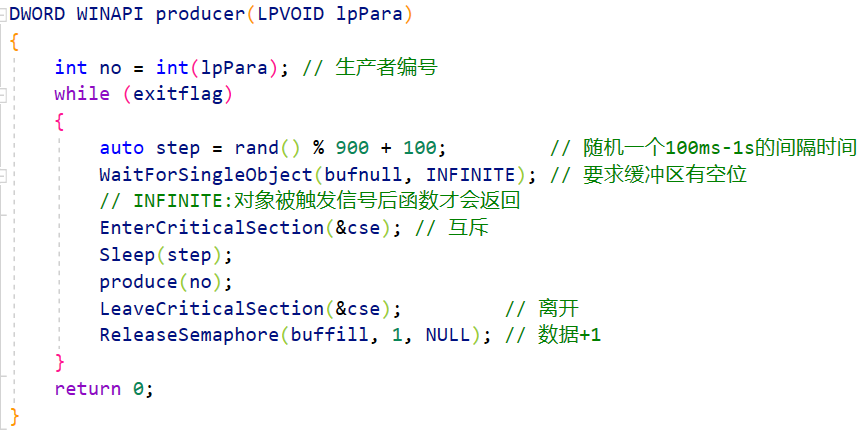
首先定义缓冲区长度、生产者个数、消费者个数、休眠时间等全局变量，并声明函数



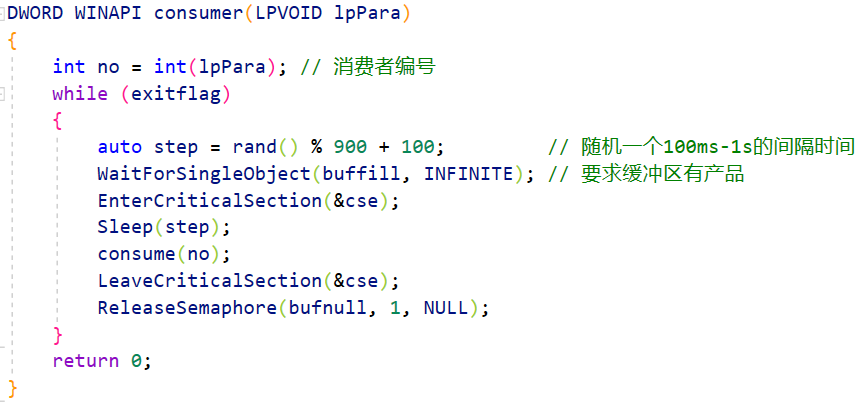
编写main函数。先使用CreateSemaphore()函数创建信号量 buffill 和 bufnull 分别表示缓冲区已有数据个数和缓冲区的空位个数，并设置信号量的初始值和最大值。然后使用InitalizeCriticalSection() 函数初始化临界区变量 cse 用于缓冲区互斥使用。接着建立生产者和消费者的线程信息变量，再使用 CreateThead()函数分别创建生产者线程和消费者线程。最后设置程序终止方式，当用户输入任意字符时终止程序，否则程序一直执行。



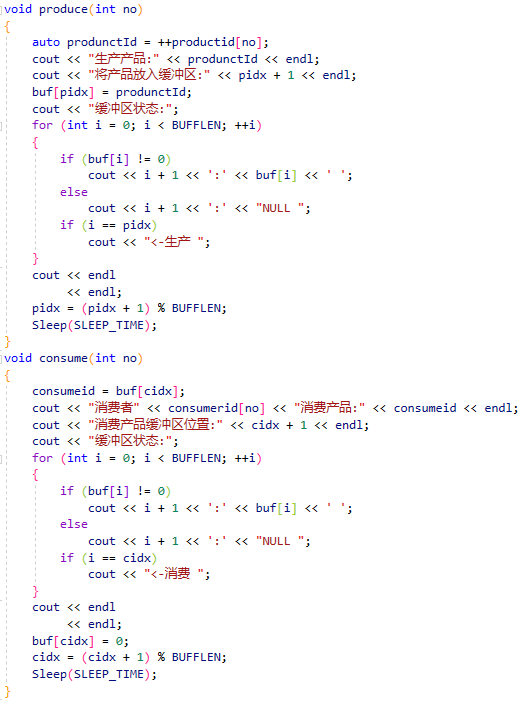
编写生产者函数。首先根据线程号设定生产者编号，由此产生符合的产品编号。接着设置随机的间隔时间，然后判断缓冲是否有空位，有才执行生产操作。使用EnterCriticalSection()和LeaveCriticalSection()函数实现共享资源操作的互斥，通过限制有且只有一个函数进入CRITICAL\_SECTION变量来实现代码段同步。具体来说，对于同一个 CRITICAL\_SECTION，当一个线程执行了EnterCriticalSection()而没有执行LeaveCriticalSection()的时候，其它任何一个线程都无法完全执行EnterCriticalSection()而不得不处于等待状态，从而避免了两个线程同时对共享资源操作的情况。最后使用ReleaseSemaphore()函数将指定的生产信号灯对象的计数增加1，用来标志缓冲区的填充状态。



同理完成消费者函数的编写

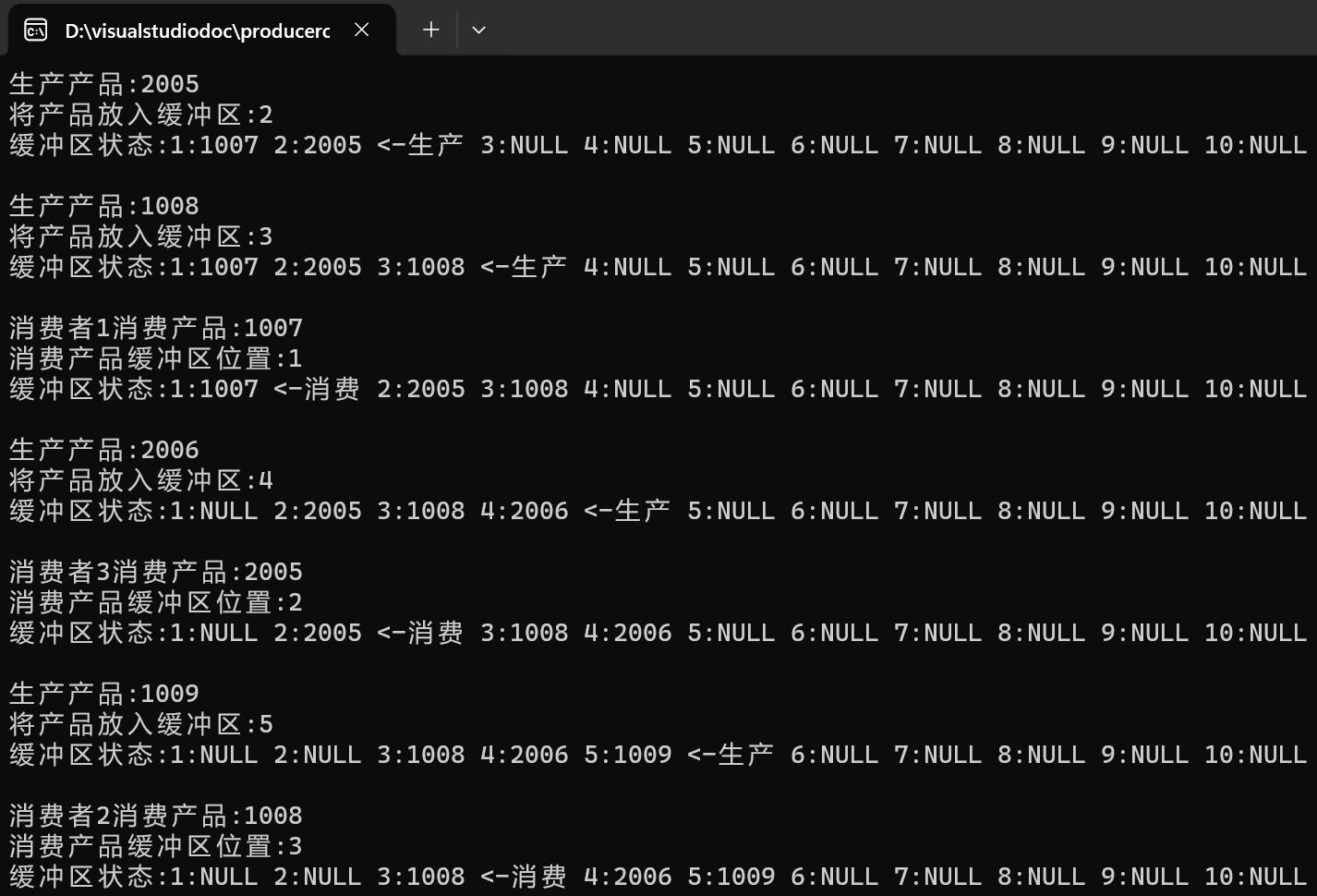


接着编写生产函数。首先生成对应的产品编号，1号生产者生产1000-1999号产品，2号生产者生产2000-2999号产品。然后使用模数实现缓冲区的循环队列，遍历缓冲区找到合适的放入位置放入产品后休眠。打印相关信息。消费函数同理。



3）测试程序

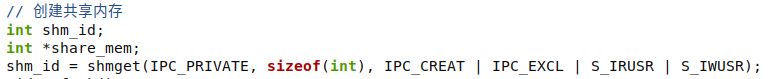
生成解决方案，调试运行结果如下。生产者和消费者线程都能按照要求运行。



**3.5 Linux信号机制实现进程通信**

1）共享内存实现

定义整型变量shmid用于保存共享内存的id，通过shmget()函数创建共享内存，参数IPC\_CREATE | IPC\_EXCL设置每次创建的都是新的一块内存，S\_IRUSR | S\_IWUSR设置可以对共享内存进行读写



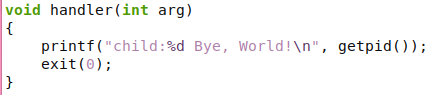
2）父进程

父进程通过shmat()函数获取共享内存指针，通过指针访问共享内存读取子进程pid，之后断开共享内存链接并且关闭共享内存。然后进入死循环，每隔两秒询问用户是否终止子进程，同时发送SIGTOP信号暂时暂停子进程，避免子进程与父进程一同输出导致信息混乱。如果用户选择终止子进程，则发送SIGUSR1信号终止子进程并跳出循环；如果用户选择不关闭，则发送SIGCONT信号让子进程继续运行。



3）子进程

子进程通过signal函数接受父进程发送的的信号SIGUSR1，并自定义接受到信号后的操作为执行handler函数，打印信息后终止进程。

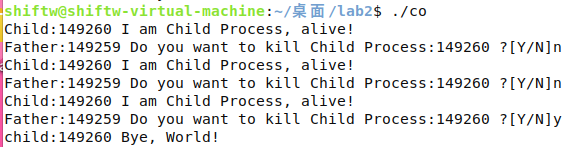


子进程同样通过shmat()函数获得共享内存的指针，由指针访问共享内存通过getpid()函数将子进程的pid保存至共享内存中，父进程可由此获得pid，接着子进程断开共享连接，进入死循环每隔两秒打印一次存活信息。



4）运行程序

编译运行程序，可以看到子进程打印信息，父进程询问用户操作时子进程的活动会中止，直到用户输入后才继续执行下一步操作。

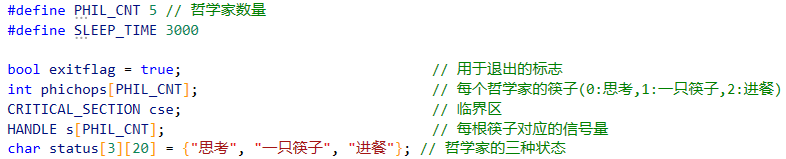


**3.6 Windows下模拟哲学家就餐**

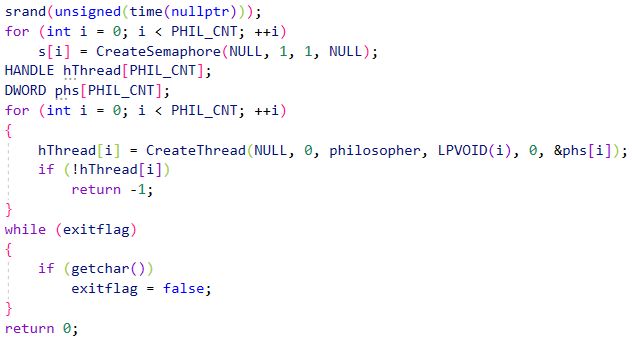
1）死锁解法

在Windows11的Visual Studio2022平台上进行开发。

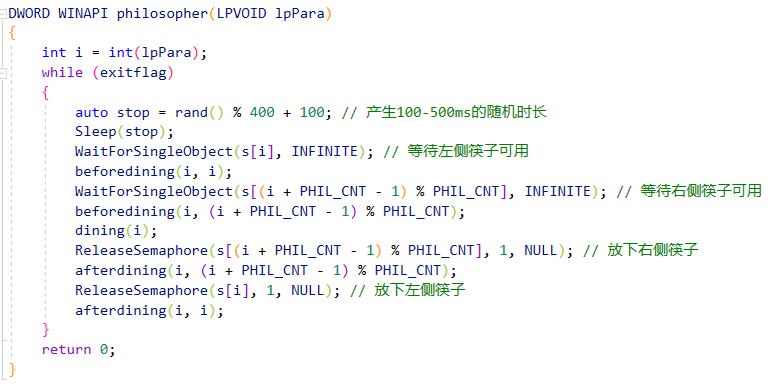
首先定义全局变量，包括哲学家数量、休眠时长、哲学家当前筷子数等等。



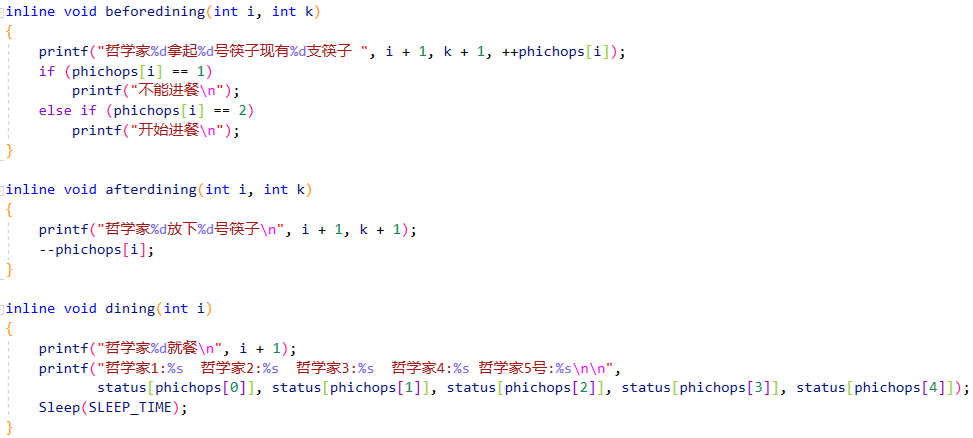
编写主函数。首先初始化随机函数，以便每次运行都能产生随机数以生成随机间隔。然后使用CreateSemaphore()函数创建每个哲学家的筷子数的信号量，接着创建每个哲学家的线程。最后构建终止程序的方法，当用户输入任意字符即终止程序，否则继续执行。



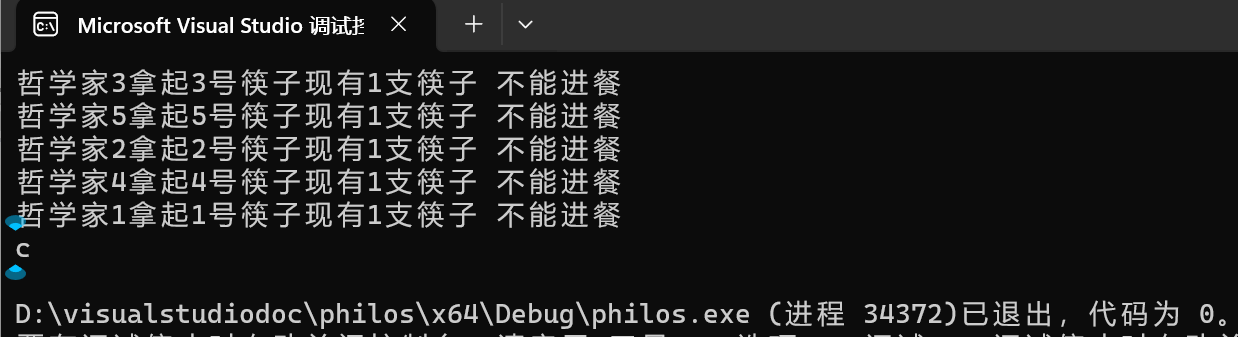
接着编写哲学家函数philosopher()，模拟哲学家就餐的过程。具体来说，每次先产生一个100-500ms的随机时间间隔，然后使用WaitForSingleObject()函数监视左侧筷子，直到左侧筷子空闲才进行就餐准备。接着继续监视右侧筷子，直到右侧筷子也空闲，执行就餐准备之后正式就餐。之后便是逆过程，释放筷子并结束用餐



接着分别编写哲学家拿筷子、释放筷子、就餐三个函数，并在过程中输出状态信息，具体函数编写不难，如下所示



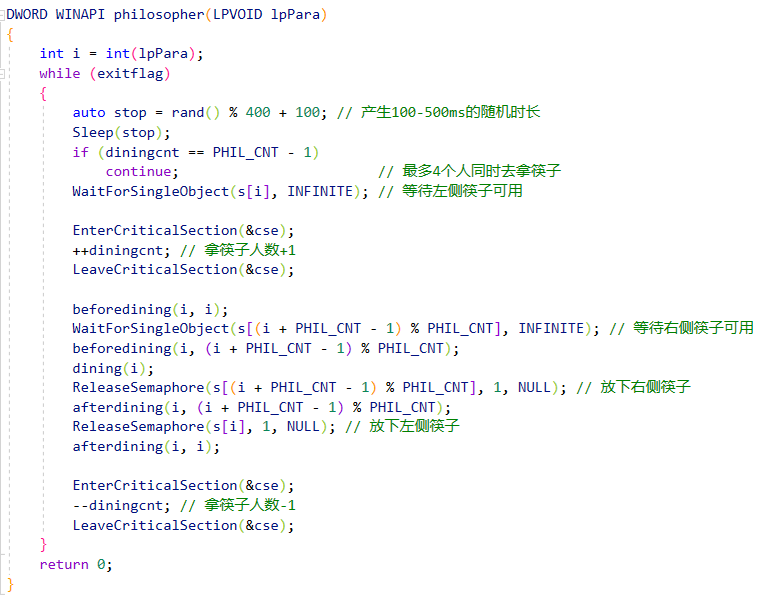
Windows11下在Visual Studio2022中生成解决方案，调试运行结果如下，观察到死锁立刻发生，没有哲学家能进餐



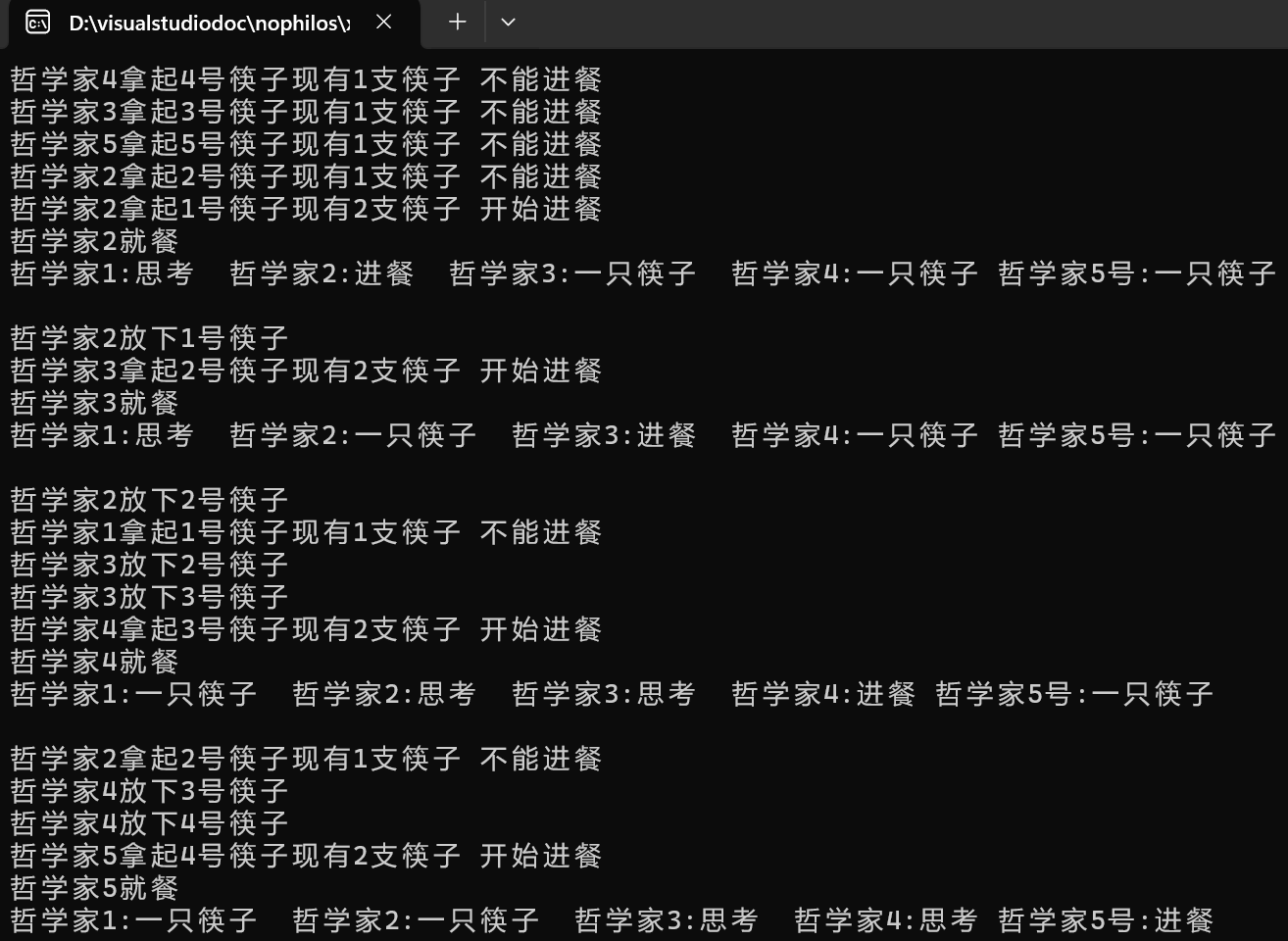
2）非死锁解法

要避免产生死锁，可以通过避免产生环路实现，也就是最多允许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，就避免给了死锁情况的产生。

故需要设置一个新变量记录拿筷子的人数，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，以确保不同线程对其访问都是互斥的，故对比死锁情况下，只需要修改哲学家函数，以及在main函数中添加初始化临界值语句。



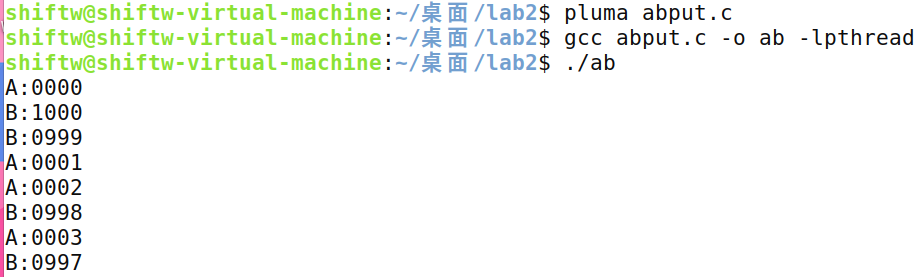
生成解决方案，调试运行，可见哲学家们总有人可以进餐，死锁情况没有发生

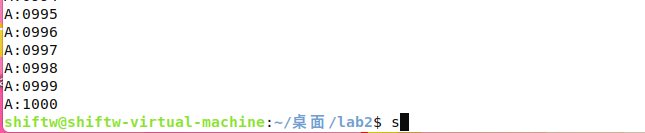


**四、实验结果**

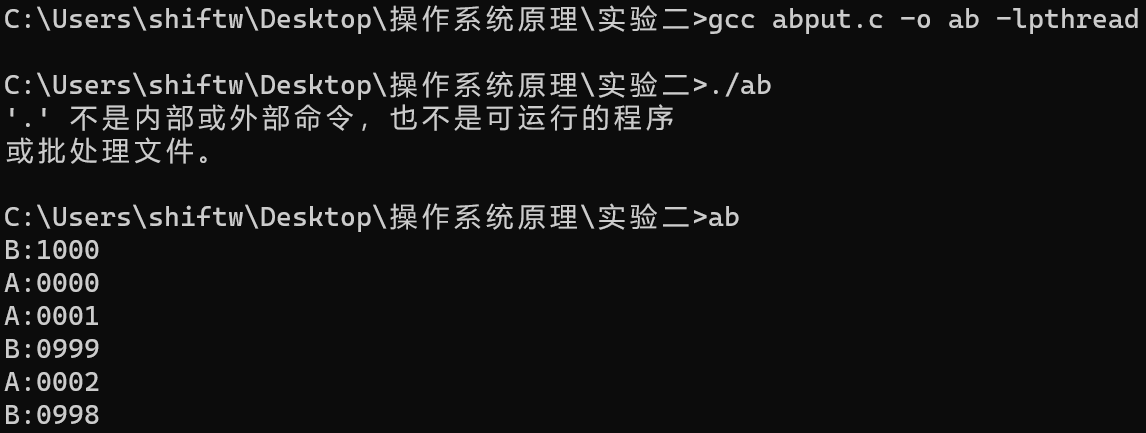
**4.1 双线程循环输出数据**

Linux下编译运行c程序，发现A和B线程循环输出，但是两者并非严格交替输出，当执行一段时间之后，可明显观察到某一个进程进度快于另一个进程，最终一个进程先行完成所有输出，只留下一个进程单独执行输出数据的操作。





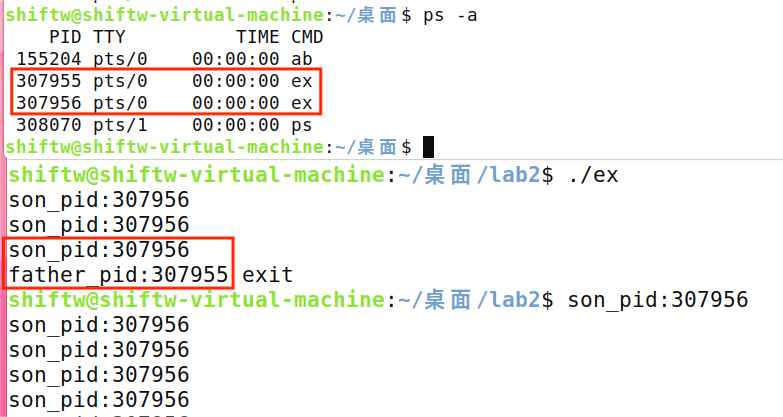
Windows11下同样编译运行c程序，结果和Linux下的一致



**4.2 Linux实验wait/exit函数**

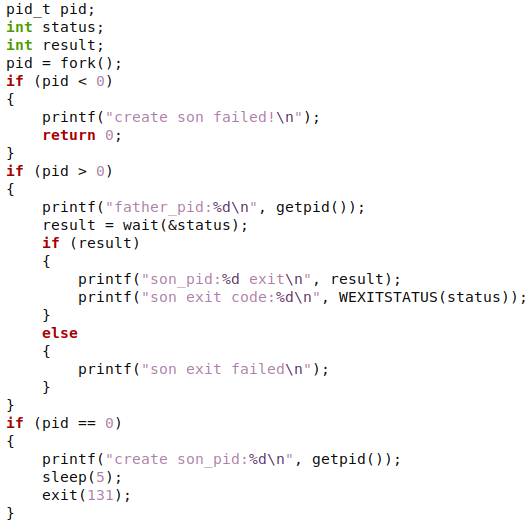
1）父进程不用wait函数

Linux下编译运行程序，同时使用ps命令显示进程，观察到显示运行的进程与输出的子进程和父进程ID一致，父进程结束后子进程单独运行，继续打印进程信息。

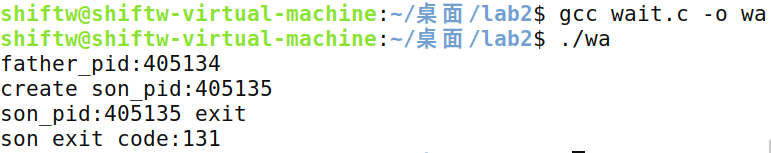


2）父进程用wait函数

创建子进程之后，父进程不使用sleep休眠，而是用wait函数等待回收子进程。子进程休眠5秒后结束，exit()设置特定的返回参数131待验证。

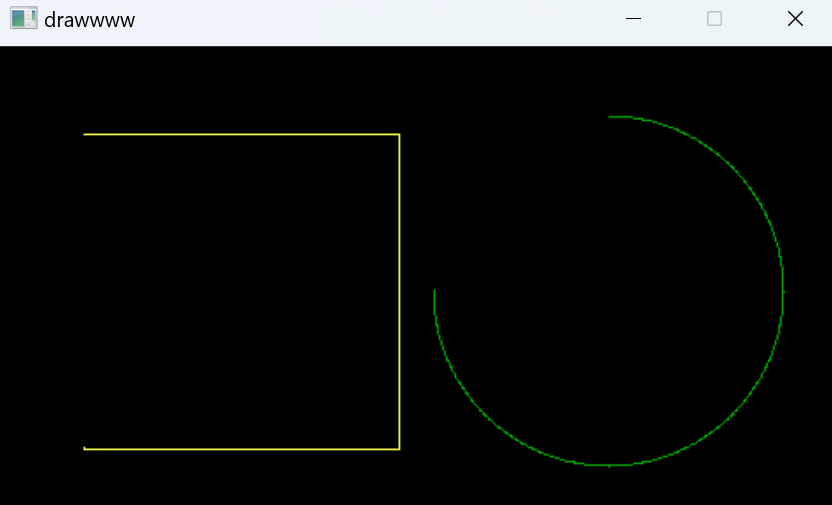


编译运行程序，观察到五秒后子进程结束，父进程正确获取子进程返回参数131并输出



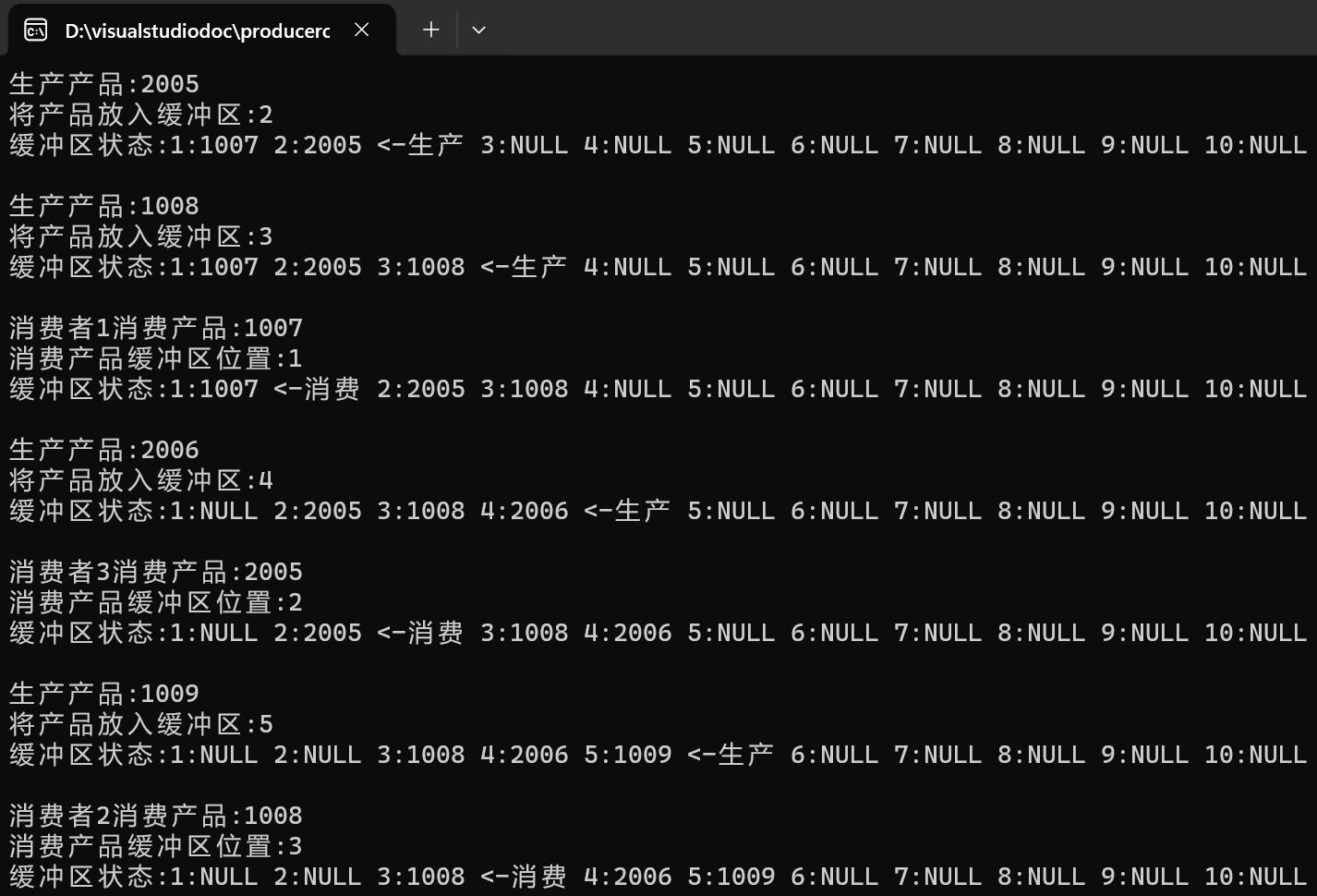
**4.3 Windows并发线程画圆画方**

在Windows11下的Visual Studio2022生成项目解决方案，运行结果如下，可观察到窗口中画圆和画方同时顺时针进行，且两者进度统一



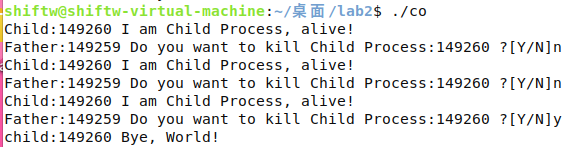
**4.4 Windows“生产者-消费者”同步控制**

在Windows11下的Visual Studio2022生成项目解决方案，运行结果如下，可观察到生产者和消费者都能按照要求运行，只有当缓冲区空余的时候才进行生产，且只有当缓冲区有产品时才消费



**4.5 Linux信号机制实现进程通信**

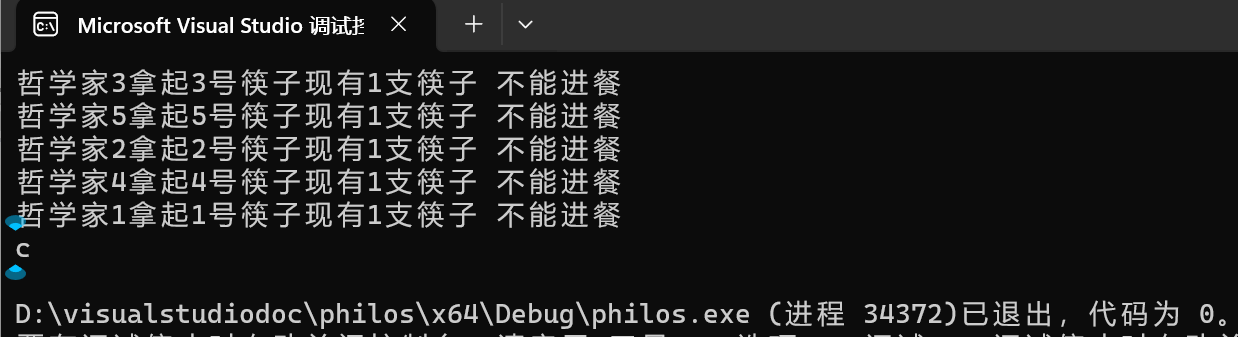
Linux下编译运行程序，可以看到子进程打印信息，父进程询问用户操作时子进程的活动会中止，直到用户输入后才继续执行下一步操作。



**4.6 Windows下模拟哲学家就餐**

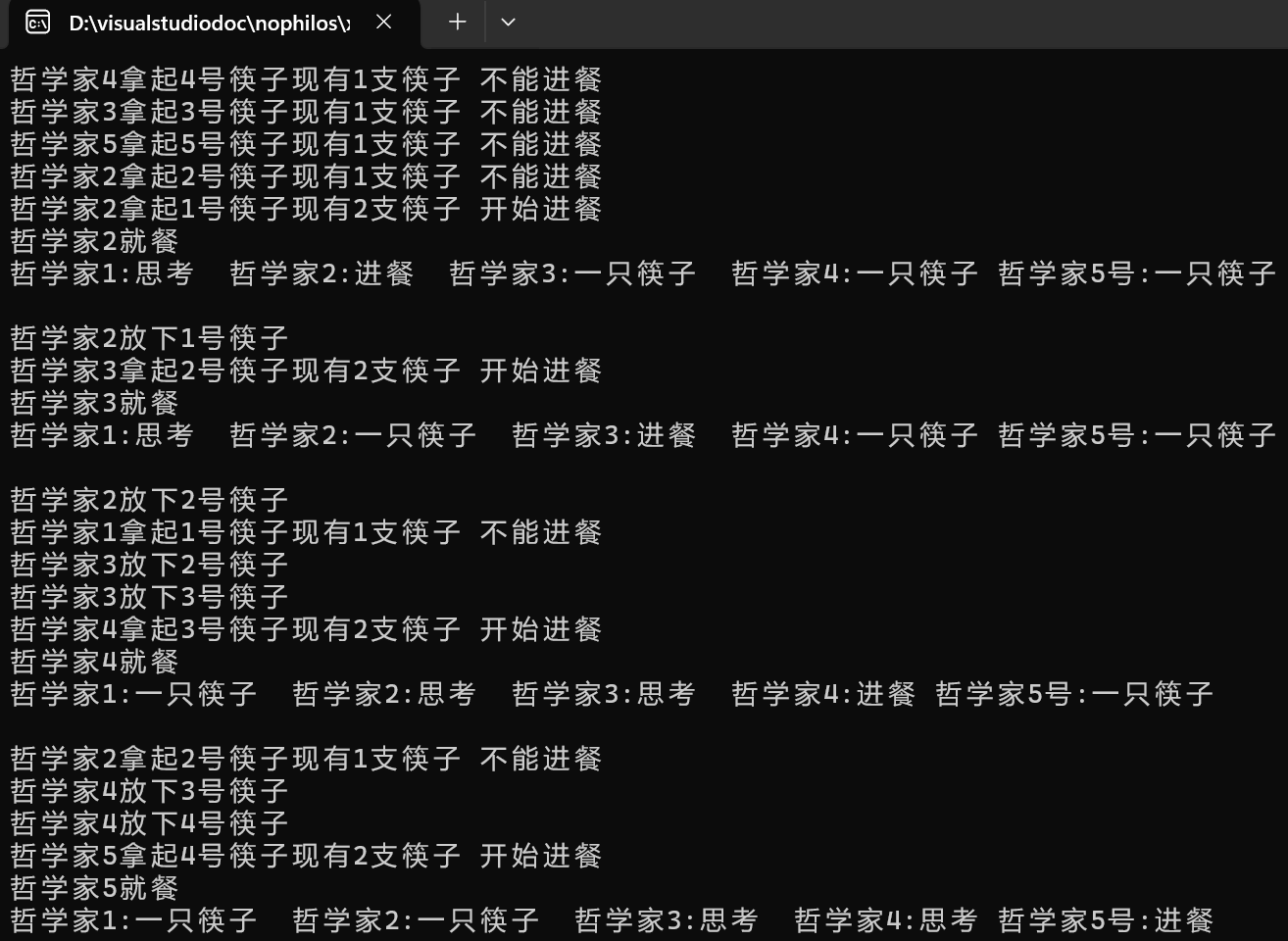
1）死锁解法

Windows11下在Visual Studio2022中生成解决方案，调试运行结果如下，观察到死锁立刻发生，没有哲学家能进餐



2）非死锁解法

采用最多允许4个哲学家拿筷子的方式避免环路产生，从而避免死锁，程序运行结果如下所示，可观察到死锁没有发生，每个阶段总有一名哲学家能够进餐



**五、实验错误排查和解决方法**

**5.1 双线程循环输出数据**

1. 使用gcc ab.c -o ab命令编译编写好的c代码报错。原因是使用了pthread库，需要加上参数-lphread才能顺利编译。

2. 双线程休眠时间太短，运行速度大差异大，难以捕获初始打印信息。解决方法是延长sleep休眠时间为1秒。

**5.2 Linux实验wait/exit函数**

1. 父进程设定的结束时间太快，没来得及在另一个终端使用ps命令查看进程父进程便终止。解决方案是延长父进程的存活时间，以便于观察。

**5.3 Windows并发线程画圆画方**

1. 无c程序画图经历和知识。上网搜索发现只需要引用graphics.h库，且添加库的配置操作十分简单，只需要下载EasyX安装程序运行即可。且画图形可以抽象为画若干个点的过程，依赖通俗易懂的官方文档的帮助，最终实现了画图函数。

2. 一开始对于如何绘制圆形理不清楚思路，后来参考了网上的样例，以( cos(-pi / 2 + ( (i \* pi) / 360) ), sin( -pi / 2 + ((i \* pi) / 360 )为变换坐标点的基础公式。

**5.4 Windows“生产者-消费者”同步控制**

1. 多次运行发现结果基本相同。原因是运行时没有初始化随机数种子，导致随机函数每次喂的种子相同，由此产生的时间间隔总是相同。只需要在开始前将当前时间作为种子初始化随机函数，即可实现每次运行的随机结果。

**5.5 Linux信号机制实现进程通信**

1. 一开始程序无法运行，检查后发现是父进程在访问共享内存的时候没有休眠，此时子进程还未将pid传入共享内存导致运行错误，应该休眠一段时间，等待子进程至少执行一轮循环之后再执行操作。

2. 用户在选择终止进程之后父进程和子进程均无输出和动作。原因是父进程在询问用户操作之前中止了子进程，以防止输出紊乱，但是在用户选择终止子进程之后，并没有恢复子进程的执行，导致即使发送了自定义终止信号，处于休眠态的子进程无法被唤醒执行相应的操作。正确的做法应该是在用户做出终止决策之后，先恢复子进程的执行再发送终止信号，这样子进程便能接收父进程的信号并做出反馈。

**5.6 Windows模拟哲学家就餐**

1. 非死锁解法在使用临界值对象后调试报错。原因是没有进行临界值初始化操作，之后的进入和释放操作是非法的，在main函数中添加InitializeCriticalSection(&cse);语句即可

**六、实验参考资料和网址**

**（1）教学课件**

**（2）https://blog.csdn.net/weixin\_44518102/article/details/124622003**

**（3）https://blog.csdn.net/m0\_47988201/article/details/116332597**

**（4）https://blog.csdn.net/low5252/article/details/104800671**

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

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

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

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

**（9）https://www.cnblogs.com/frisk/p/11602973.html**