中国地质大学(武汉)实验教学系列教材
中国地质大学(武汉)实验技术研究项目资助

操作系统原理实习指导书

CAOZUO XITONG YUANLI SHIXI ZHIDAOSHU

谷淑化
张　霞
张求明
常　虹
◎编著

中国地质大学出版社有限责任公司
ZHONGGUO DIZHI DAXUE CHUBANSHE YOUXIAN ZEREN GONGSI

操作系统原理实习指导书

CAOZUO XITONG YUANLI SHIXI ZHIDAOSHU

谷淑化　张　霞　张求明　常　虹　编著

中国地质大学出版社有限责任公司
ZHONGGUO DIZHI DAXUE CHUBANSHE YOUXIAN ZEREN GONGSI

图书在版编目(CIP)数据

操作系统原理实习指导书/谷淑化,张霞,张求明,常虹编著. —武汉:中国地质大学出版社有限责任公司,2011.11

ISBN 978-7-5625-2743-5

Ⅰ.操…
Ⅱ.①谷…②张…③张…④常…
Ⅲ.操作系统-高等学校-教材
Ⅳ.TP316

中国版本图书馆 CIP 数据核字(2011)第 211464 号

操作系统原理实习指导书 谷淑化 张 霞 张求明 常 虹 编著

责任编辑:胡珞兰 责任校对:张咏梅

出版发行:中国地质大学出版社有限责任公司(武汉市洪山区鲁磨路 388 号) 邮政编码:430074

电 话:(027)67883511 传 真:67883580 E-mail:cbb@cug.edu.cn

经 销:全国新华书店 http://www.cugp.cug.edu.cn

开本:787 毫米×1 092 毫米 1/16 字数:125 千字 印张:4.875

版次:2011 年 11 月第 1 版 印次:2011 年 11 月第 1 次印刷

印刷:教文印刷厂 印数:1—1 000 册

ISBN 978-7-5625-2743-5 定价:12.00 元

前 言

操作系统是计算机软件系统的核心，是所有计算机系统的基础和支撑，它管理和控制着计算机系统中的软硬件资源，并为用户使用计算机提供一个方便灵活、安全可靠的工作环境，可以说操作系统是计算机系统的灵魂。本书遵循操作系统原理课程的教学大纲的要求，针对计算机相关专业本科生的专业定位和人才培养目标而编写。

由于操作系统原理过于抽象，要真正理解操作系统的概念，必须将原理与实践相结合。操作系统原理课程的实验环节一直是该课程的难点。本书通过 Windows 操作系统的编程接口，提供了一些编程实例，使学生对操作系统有一个明确清晰的认识，在程序设计方面也得到训练。

本书由 5 章组成，建议课堂讲授 44 学时，实验 20 学时。

第 1 章　进程管理。主要介绍程序的并发执行，进程的概念、状态及其转换，进程同步与互斥的概念及各种实现策略。建议实验 6 学时。

第 2 章　资源管理。主要介绍进程调度算法，死锁的概念、死锁的预防与避免以及银行家算法。建议实验 4 学时。

第 3 章　存储管理。主要介绍存储管理的概念和管理的目的，存储管理的四大基本功能——内存分配与回收、逻辑地址到物理地址的转换、存储保护和内存扩充，实存管理和虚存管理的各种策略。本章重点是各页面置换算法的实现。建议实验 4 学时。

第 4 章　文件管理。主要介绍文件系统的概念、文件的组织结构及存取方法、文件存储空间的管理、文件的安全与保护。建议实验 4 学时。

第 5 章　磁盘管理。主要介绍磁盘调度的概念。通过磁盘调度算法模拟设计，掌握各种磁盘调度算法的思想。建议实验 2 学时。

本书每章由实验目的、预备知识和实验内容组成，并配有大量习题及参考答案。

本书第 1 章至第 5 章由谷淑化编写，附录由张霞编写，常虹参与了本书的资

料整理、编辑修订等工作，全书由张求明、童恒建统稿，并提出了宝贵意见。教材编写委员会和中国地质大学出版社有限责任公司在本书的出版过程中给予了大力支持。在此一并表示衷心地感谢！

由于作者水平有限，书中存在疏漏与错误在所难免，敬请读者批评指正。

作者

2011 年 8 月 1 日

目　录

第1章 进程管理

1.1 进程互斥

1.1.1 实验目的

(1)加深对进程概念的理解,明确进程和程序的区别。进一步认识并发执行的实质。

(2)分析进程竞争资源的现象,学习解决进程互斥的方法。

(3)通过调试和编写进程调度程序,加深对进程相关概念和进程调度的理解。

1.1.2 预备知识

1.1.2.1 进程的定义

进程的概念是20世纪60年代初期,首先由麻省理工学院的MULTICS系统和IBM公司的TSS/360系统引入的。其后,有许多学者对进程下过各种定义。下面,仅列举几种比较能反映进程实质的定义。

·行为的一系列规则叫做程序,程序在处理机上执行时所发生的活动称为进程(Dijkstra)。

·进程是这样的计算部分,它是可以和其他计算并行的一个计算(Donovan)。

·进程(有时称为任务)是一个程序与其数据一道通过处理机的执行所发生的活动(Alan. C. Shaw)。

·进程是执行中的程序(Ken Thompson and Dennis Ritchie)。

·进程,即是一个具有一定独立功能的程序关于某个数据集合的一次活动。

1.1.2.2 进程与程序的区别

·程序是指令的集合,是静态的概念。进程是程序在处理机上的一次执行的过程,是动态的概念。程序可以作为软件资料长期保存。进程是有生命周期的。

·进程是一个独立的运行单位,能与其他进程并行(并发)活动。而程序则不能。

·进程是竞争计算机系统有限资源的基本单位,也是进行处理机调度的基本单位。

·一个程序可以作为多个进程的运行程序,一个进程也可以运行多个程序。

1.1.2.3 进程的状态

进程的基本状态包括就绪状态、运行状态和等待状态。

·就绪状态(Ready)。存在于处理机调度队列中的那些进程,它们已经准备就绪,一旦得到CPU,就立即可以运行,这些进程所取得的状态为就绪状态(有多个进程处于此状态)。

· 运行状态(Running)。当进程由调度/分派程序分派后,得到 CPU 控制权,它的程序正在运行,该进程所处的状态为运行状态(在系统中,总只有一个进程处于此状态)。

· 等待状态(Wait)。若一个进程正在等待某个事件的发生(如等待 I/O 的完成),而暂停执行,这时,即使给它 CPU 时间,它也无法执行,则称该进程处于等待状态。

1.1.2.4 进程控制块

存放进程的管理和控制信息的数据结构称为进程控制块。它是进程管理和控制的最重要的数据结构,在创建时,建立 PCB,并伴随进程运行的全过程,直到进程撤消而撤消。PCB 就像我们的户口。

1.1.2.5 进程互斥

在操作系统中,当某一进程正在访问某临界区时,就不允许其他进程进入,否则就会发生(后果)无法估计的错误。我们把进程之间的这种相互制约的关系称为进程互斥。

1.1.2.6 Windows 下的系统调用

Windows 所创建的每个进程都从调用 Create Process()API 函数开始,该函数的任务是在对象管理器子系统内初始化进程对象。每一个进程都以调用 Exit Process 或 Terminate Process()API 函数终止。通常应用程序的框架负责调用 Exit Process()函数。对于 C++运行库来说,这一调用发生在应用程序的 main()函数返回之后。

· 创建进程。Create Process()函数调用的核心参数时刻执行文件运行时的文件名及其命令性。表 1.1 详细地列出了每个参数的类型和名称。

表 1.1 Create Process()函数的参数

参数名称	使用目的
LPCTSTR lp Applivation Name	全部或部分的知名包括可执行的 exe 文件的文件名
LPCTSTR lp Command Line	向可执行文件发送的参数
LPSECURIITY_ATTRIBUTES Lp Process Attributes	返回进程句柄的安全属性。主要指明这一句柄是否由其他进程所继承
LPSECURIITY_ATTRIBUTES Lp Thread Attributes	返回进程的主线程的句柄的安全属性
BOOL b Inherit Handle	一种标志,告诉系统允许新进程继承创建者进程的句柄
DWORD dw Creation Flage	特殊的创建标志(如 CREATE_SUSPENDED)的位标记
LPVOID lp Environment	向新进程发送的一套环境变量,如为 null 值则发送调用者环境
LPCTSTR lp Current Directory	新进程的启动目录
STARTUPINFO lp Startup Info	STARTUPINFO 结构,包括新进程的输入和输出配置的详情
LPPROCESS_INFORMATION Lp Process Information	调用的结果块;发送新应用程序的进程和主线程的句柄和 ID

可以指定第 1 个参数,即应用程序的名称,其中包括相对于当前进程的当前目录的全路径或者利用搜索方法找到的路径;lp Command Line 参数允许调用者向新应用程序发送数据;接下来的 3 个参数与进程和它的主线程以及返回的指向该对象的句柄的安全性有关。

然后是标志参数,用以在 dw Creation Flags 参数中指明系统应该给予新进程什么行为。经常使用的标志是 CREATE_SUSPNDED,告诉主线程立即暂停。当准备好时,应该使用

Resume Thread()API 来启动进程。另一个常用的标志是 CREATE_NEW_CONSOLE,告诉新进程启动自己的控制台窗口,而不是利用父窗口。这一参数还允许设置进程的优先级,用以向系统指明,相对于系统中所有其他活动进程来说,给此进程多少 CPU 时间。

接着是 Create Process()函数调用所需要的 3 个通常使用默认值的参数。第 1 个参数是 lp Environment 参数,指明为新进程提供的环境;第 2 个参数是 lp Current Diredtory,可用于向主创进程发送与默认目录不同的新进程使用的特殊的当前目录;第 3 个参数是 STARTUPINFO 数据结构所必需的,用于在必要时知名新应用程序的主窗口的外观。

Create Process()函数的最后一个参数是用于新进程对象及其主线程的句柄和 ID 的返回值缓冲区。以 PROCESS_INFORMATION 结构中返回的句柄调用 Close Handle()API 函数是重要的,因为如果不将这些句柄关闭,有可能危及进程终止之前任何未释放的资源。

· 正在运行的进程。

如果一个进程拥有至少一个执行线程,则为正在系统中运行的进程。通常,这种进程存在。当主线程结束时,调用 Exit Process()API 函数,通知系统终止它所拥有的所有正在运行、准备运行或正在挂起的其他线程。当进程正在运行时,可以查看它的许多特性,其中少数特性也允许加以修改。

首先可查看的进程特性是系统进程标识符(PID),可利用 Get Current Process Id()API 函数来查看,与 Get Current Process()相似,对该函数的调用性不能失败,但返回的 PID 在整个系统中都可使用。其他的可显示当前进程信息的 API 函数还有 Get Startup Info()和 Get Process Shutdown Parameters(),可给出进程存活期内的配置详情。

通常,一个进程需要它的运行期环境的信息。例如 API 函数 Get Module File Name()和 Get Command Line()可以给出用在 Create Process()中的参数以启动应用程序。在创建应用程序时刻使用的另一个 API 函数是 Is Debugger Present()。

可利用 API 函数 Get Cui Resources()来查看进程的 GUI 资源。此函数即可返回指定进程中的打开的 GUI 对象的数目,也可返回指定进程中打开的 USER 对象的数目。进程的其他性能信息可通过 Get Process Io Counters()、Get Process Priority Boost()、Get Process Times()和 Get Process Working Set Sise()API 得到。以上这几个 API 函数都只需要具有 PROCESS_QUERY_INFORMATION 访问权限的指向所感兴趣进程的句柄。

另一个可用进程信息查询的 API 函数是 Get Process Version()。此函数只需感兴趣进程的 PID(进程标识号)。这一 API 函数与 Get Version Ex()共同作用,可确定运行进程的系统的版本号。

· 终止进程。

所有进程都是以调用 Exit Process()或者 Terminate Process()函数结束的。但最好使用前者而不要使用后者,因为进程是在完成了它的所有的关闭"职责"之后以正常的终止方式来调用前者的。而外部进程通常调用后者即突然终止进程的进行,由于关闭时的途径不太正常,有可能引起错误的行为。

Terminate Process()API 函数只要打开带有 PROCESS_TERMINATE 访问权的进程对象,就可以终止进程,并向系统返回指定的代码。这只是一种"野蛮"的终止进程的方式,但有时却是需要的。

如果开发人员确实有机会来设计"谋杀"(终止别的进程的进程)和"受害"进程(被终止的

进程)时,应该创建一个进程间通信的内核对象——一个互斥程序,这样一来,"受害"进程只在等待或周期性地测试它是否应该终止。

· 等待函数。

使用等待函数既可以保证线程的同步,又可以提高程序的运行效率。最常用的等待函数是 WaitForSingleObject,该函数的声明如下:

DWORDWaitForSingleObject (HANDLE h Handle, DWORD dwMilliseconds);

参数 hHandle 是同步对象的句柄。参数 dwMilliseconds 是以毫秒为单位的时间间隔,如果该参数为 0,那么函数就测试同步对象的状态并立即返回,如果该参数为 INFINITE,则超时间隔是无限的。

WaitForSingleObject 的返回值如表 1.2 所示。

表 1.2 WaitForSingleObject 返回值

返回值	含 义
WAIT_FAILED	函数失败
WAIT_OBJECT_0	指定的同步对象处于有信号的状态
WAIT_ABANDONED	拥有一个 mutex 的线路已经中断了,但未释放该 mutex
WAIT_TIMEOUT	超时返回,并且同步对象无信号

1.1.3 实验内容

1.1.3.1 进程的创建

【实验 1】 阅读下面源程序,完成实验任务。利用系统调用 fork()创建两个子进程,当此程序运行时,在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符:父进程显示字符"a";子进程分别显示字符"b"和字符"c"。

```
#include<stdio. h>
main ( )
{
        int, p2;
        while ((p1=fork ( )) ==- 1)              /* 创建子进程 p1 */
        if (p1==0)     put char ('b');
        else
                {
                        while((p2=fork())==- 1);     /* 创建子进程 p2 */
                        if (p2==0)     putchar ('c');
                        else       put char ('a')
                        }
}
```

实习任务：

(1)多次运行程序，分析为什么有不同的执行结果。

(2)模仿上述程序，编程使用 fork()创建一个子进程，要求在父、子进程中显示出 fork()的返回值及父、子进程的唯一 pid 值。

1.1.3.2　进程的控制

【实验 2】　修改已编写的程序，将每一个进程输出一个字符改为每一个进程输出一句话。阅读源程序，完成实验任务。

```
#include<stdio. h>
main ( )
{
    int p1, p2, I;
    while ((p1=fork ( )) ==- 1)                    /* 创建子进程 p1 */
    if (p1==0)
        for( i=0;i<10;i++)
            printf ("daughter % d\ n",i)
    else
      {
        while((p2=fork( ))==- 1);      /* 创建子进程 p2 */
        if (p2==0)
            for (i=0; i<10; i++)
                printf ("son % d\ n", i)
        else
            for (i=0; i<10; i++)

                printf ("parent % d\ n", i)
      }
  }
```

实验任务：

(1)多次运行程序，分析为什么有不同的执行结果。

(2)将 for(i=0;i<10;i++)改为 for(i=0;i<100;i++)并分析执行结果的变化。

【实验 3】　下面的程序演示 fork( )和 exec( )联合使用的情况。阅读源程序，完成实验任务。

```
#include<stdio.好》
#include<unistd. H>
main ( )
{
    int pid;
    pid =fork ( )                          /* 创建子进程 */
    switch (pid)
```

```
        {
            case    -1;                    /* 子进程 */
                    printf ("fork fail! \n");
                    exit (1)
            case     0;                    /* 子进程 */
                    if (execl ("/bin/ls"," -1"," -color", NULL) <0
                     {printf ("exec fail! \n");
                     exit (1)
                     }
            default:                       /* 父进程 */
                     wait(NULL);    /* 同步 */
                     printf ("ls completed! \n")
        }
    }
```

实验任务：

(1)写出执行结果第 1 行：(按倒序)列出当前目录下所有文件和子目录。

(2)写出 s －1 的结果，最后一行，与(1)中执行结果的第 1 行比较。

1.1.3.3　文件的加锁、解锁

【实验 4】　下面的程序演示了系统调用 lockf()的使用。阅读源程序，完成实验任务。

```
#include<stdio.h>
#include<unistd.h>
Main ()
{
int p1, p2, I;
FILE * fp
fp=fopen ("to_be_locked. Txt","w+")
if (fp==NULL)
    {
     printf ("Fail to creat file")
     exit ( - 1)
    }
            while((p1=fork( ))==- 1)        /* 创建子进程 p1 */
if (p1==0)
    {
     lockf((int),1,0);                  /* 加锁 */
     for (i=0; I<10; i++)
     fprintf (fp,"daughter % d\ n" ,i)
     lock f((int)分配,0,0)          /* 解锁 */
    }
```

```
else
 {
        while ((p2=fork( ))==-1)        /* 创建子进程 p2 */
  if (p2==0)
  {
   lockf((int)fp,1,0);                  /* 加锁 */
   for (i=0;i<10;i++)
   fprintf (fp,"son % d\ n", i)
   lockf((int)fp,0,0)                   /* 解锁 */
  }
 else
   {
    wait (NULL)
    locdf((int)fp,1,0)                   /* 加锁 */
    for (i=0; i<10; i++)
    fprintf (fp,"parent % d\ n", i)
    lockf((int)fp,0,0);                  /* 解锁 */
   }
  }
  fclose9分配)
  }
```

程序编译、运行后执行下面的 cat 命令：

cat to_be_locked.txt

实验任务：写出 cat 命令的执行结果。

1.1.3.4　应用实例

问题描述：有父亲、儿子、女儿 3 人和一个盘子。当盘子空时，父亲往盘中放水果（只有苹果和香蕉），每次随机放苹果或香蕉，儿子和女儿马上取该水果，但是儿子只从盘中拿苹果，女儿只从盘中拿香蕉。请模拟这样的一个过程，其中父亲放水果 20 次。

程序分析：

首先，父亲放一个苹果在盘，儿子马上取苹果这件事与父亲放一个香蕉，女儿马上取之这件事是互斥的。即在程序中体现在：当父亲每次放水果之后马上唤醒儿子或女儿来取，当一个孩子还没有拿到水果之前，父亲不允许放水果，另外一个孩子也不许来破坏。我们用“临界区”区实现：

父亲函数：

```
……
进入临界区
放苹果或香蕉
……
```

儿子或女儿函数：

……

取苹果或香蕉

退出临界区

……

另外，儿子和女儿在默默等待，只要父亲放了水果他们就拿。程序中表现在：儿女见到盘子没有水果，就被阻塞，父亲每次放水果后唤醒阻塞的孩子函数。

请编程实现该问题。

1.2 进程同步

1.2.1 实验目的

(1)加深对进程同步的理解，学习解决进程同步的方法。

(2)通过调试和编写进程同步程序，掌握典型的进程同步问题。

1.2.2 预备知识

1.2.2.1 进程和线程

在 Windows 32 位操作系统中，所谓多任务是指系统可以同时运行多个进程，而每个进程也可以同时执行多个线程。

所谓进程就是应用程序的运行实例。每个进程都有自己私有的虚拟地址空间。每个进程都有一个主线程，但可以建立另外的线程。进程中的线程是并发执行的，每个线程占用 CPU 的时间由系统来划分。我们可以把线程看成是操作系统分配 CPU 时间的基本实体。

进程中的所有线程共享进程的虚拟地址空间，这意味着所有线程都可以访问进程的全局变量和资源。这一方面为编程带来了方便，但另一方面也容易造成冲突。

虽然在进程中进行费时的工作不会导致系统的挂起，但这会导致进程本身的挂起。所以，如果进程既要进行长期的工作，又要响应用户的输入，那么它可以启动一个线程来专门负责费时的工作，而主线程仍然可以与用户进行交互。

1.2.2.2 线程的同步

多线程的使用会产生一些新的问题，主要是如何保证线程的同步执行。多线程应用程序需要使用同步对象和等待函数来实现同步。同步问题是实现远程数据采集或远程自动控制编程中的关键技术问题。

1.2.2.3 同步的原因

由于同一进程的所有线程共享进程的虚拟地址空间，并且线程的中断是汇编语言级的，所以可能会发生两个线程同时访问同一个对象(包括全局变量、共享资源、API 函数和 MFC 对象等)的情况，这有可能导致程序错误。例如，如果一个线程在未完成对某一大尺寸全局变量的读操作时，另一个线程又对该变量进行了写操作，那么第一个线程读入的变量值可能是一种修改过程中的不稳定值。属于不同进程的线程在同时访问同一内存区域或共享资源时，也会

存在同样的问题。因此，在多线程应用程序中，常常需要采取一些措施来同步线程的执行。

1.2.2.4　同步的方法

由于线程间需要同步机制，才能协调运行。在 Windows 32 编程中，主要提供了以下几种方法。

· 等待函数。

这些函数只有在作为其参数的一个或多个同步对象产生信号时才会返回。在超过规定的等待时间后，不管有无信号，函数也都会返回。在等待函数未返回时，线程处于等待状态，此时线程只消耗很少的 CPU 时间。

· 同步对象。

同步对象用来协调多线程的执行，它可以被多个线程共享。线程的等待函数用同步对象的句柄作为参数，同步对象应该是所有要使用的线程都能访问到的。同步对象的状态要么是有信号的，要么是无信号的。同步对象主要有 3 种：事件、mutex 和信号灯。

事件对象(Event)是最简单的同步对象，它包括有信号和无信号两种状态。在线程访问某一资源之前，也许需要等待某一事件的发生，这时用事件对象最合适。例如，只有在通信端口缓冲区收到数据后，监视线程才被激活。

mutex 对象的状态在它不被任何线程拥有时是有信号的，而当它被拥有时则是无信号的。mutex 对象很适合用来协调多个线程对共享资源的互斥访问(Mutually Exclusive)。

信号灯对象维护一个从 0 开始的计数，在计数值大于 0 时对象是有信号的，而在计数值为 0 时则是无信号的。信号灯对象可用来限制对共享资源进行访问的线程数量。线程用 Create Semaphore()函数来建立信号灯对象，在调用该函数时，可以指定对象的初始计数和最大计数。在建立信号灯时也可以为对象起个名字，别的进程中的线程可以用 Open Semaphore ()函数打开指定名字的信号灯句柄。

1.2.3　实验内容

1.2.3.1　共享存储区

【实验 1】　生产者-消费者问题

生产者-消费者问题是一个经典的进程同步问题，该问题最早由 Dijkstra 提出，用以演示他提出的信号量机制。此进程同步问题是在同一个进程地址空间内执行的两个线程。生产者线程生产物品，然后将物品放置在一个空缓冲区中供消费者线程消费。消费者线程从缓冲区中获得物品，然后释放缓冲区。当生产者线程生产物品时，如果没有空缓冲区可用，那么生产者线程必须等待消费者线程释放出一个空缓冲区。当消费者线程消费物品时，如果没有满的缓冲区，那么消费者线程将被阻塞，直到新的物品被生产出来。

下面给出了解决生产者-消费者问题的应用程序，阅读该程序并完成实验任务。

```
#include <windows.h>
#include <iostream>

const unsigned short SIZE_OF_BUFFER=10; //缓冲区长度
unsigned short ProductID=0; //产品号
unsigned short ConsumeID=0; //将被消耗的产品号
```

```
unsigned short in=0; //产品进缓冲区时的缓冲区下标
unsigned short out=0; //产品出缓冲区时的缓冲区下标

int g_buffer[SIZE_OF_BUFFER]; //缓冲区是个循环队列
bool g_continue=true; //控制程序结束
HANDLE g_hMutex; //用于线程间的互斥
HANDLE g_hFullSemaphore; //当缓冲区满时迫使生产者等待
HANDLE g_hEmptySemaphore; //当缓冲区空时迫使消费者等待

DWORD WINAPI Producer(LPVOID); //生产者线程
DWORD WINAPI Consumer(LPVOID); //消费者线程

int main()
{
//创建各个互斥信号
g_hMutex=CreateMutex(NULL,FALSE,NULL);
g_hFullSemaphore=CreateSemaphore(NULL,SIZE_OF_BUFFER - 1,SIZE_OF_BUFFER - 1,NULL);
g_hEmptySemaphore=CreateSemaphore(NULL,0,SIZE_OF_BUFFER - 1,NULL);

//调整下面的数值,可以发现,当生产者个数多于消费者个数时,
//生产速度快,生产者经常等待消费者;反之,消费者经常等待
const unsigned short PRODUCERS_COUNT=3; //生产者的个数
const unsigned short CONSUMERS_COUNT=1; //消费者的个数

//总的线程数
const unsigned short THREADS_COUNT=PRODUCERS_COUNT+ CONSUMERS_COUNT;

HANDLE hThreads[PRODUCERS_COUNT]; //各线程的 handle
DWORD producerID[CONSUMERS_COUNT]; //生产者线程的标识符
DWORD consumerID[THREADS_COUNT]; //消费者线程的标识符

//创建生产者线程
for (int i=0;i<PRODUCERS_COUNT;++i){
hThreads[i]=CreateThread(NULL,0,Producer,NULL,0,&producerID[i]);
if (hThreads[i]==NULL) return - 1;
}
//创建消费者线程
for (i=0;i<CONSUMERS_COUNT;++i){
hThreads[PRODUCERS_COUNT+i]=CreateThread(NULL,0,Consumer,NULL,0,&consumerID[i]);
```

```
if (hThreads[i]==NULL) return-1;
}

while(g_continue){
if(getchar()){ //按回车后终止程序运行
g_continue=false;
}
}

return 0;
}

//生产一个产品。简单模拟了一下,仅输出新产品的 ID 号
void Produce()
{
std::cerr<<"Producing " <<++ProductID<<" ... ";
std::cerr<<"Succeed"<<std::endl;
}

//把新生产的产品放入缓冲区
void Append()
{
std::cerr <<"Appending a product ... ";
g_buffer[in] =ProductID;
in=(in+1)% SIZE_OF_BUFFER;
std::cerr<<"Succeed"<<std::endl;

//输出缓冲区当前的状态
for (int i=0;i<SIZE_OF_BUFFER;++i){
std::cout <<i <<": " <<g_buffer[i];
if (i==in) std::cout <<" <--生产";
if (i==out) std::cout <<" <--消费";
std::cout <<std::endl;
}
}

//从缓冲区中取出一个产品
void Take()
{
```