实验二 生产者与消费者

姓名：叶倩琳

学号：201706061330

完成日期：2019/10/28

# 一、实验目的

# 利用Windows提供的API函数，编写程序，解决生产者与消费者问题，实现进程的互斥与同步。

# 二、实验内容

1、进程的互斥与同步。编写一段程序，模拟生产者和消费者线程，实现进程的互斥与同步。

2、利用VC++6.0实现上述程序设计和调试操作，对于生产者和消费者线程操作的成功与否提供一定的提示框。

3、通过阅读和分析实验程序，熟悉进程的互斥与同步的概念。

# 主要程序结构

本实验要求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。生产者线程生产物品时，若无空缓冲区可用时必须等待；消费者线程消费物品时，若缓冲区为空，消费者线程将被阻塞，直到新的物品被生产出来。因此，需要使用以下信号量：

 一个互斥信号量，保证生产者线程和消费者线程互斥使用缓冲区；

一个同步信号量，当生产者线程生产出一个物品时可以用它向消费者线程发出信号；

一个同步信号量，消费者线程释放出一个空缓冲区时可以用它向生产者线程发出信号；

**（1）首先，创建一个struct结构。**其中包含有：标志缓冲区是否为空的信号量m\_S\_Empty，标志缓冲区是否放满的信号量m\_S\_Full，一个防止生产者与消费者同时访问缓冲区的互斥信号量m\_M\_Mutex，共享缓冲区队列，一个判断生产者是否要结束生产的bool类型producerfinished。

**（2）其次，编写生产者函数和消费者函数。**生产者函数中通过for循环实现多次生产。消费者函数中通过while循环，当生产者没有结束生产（即producerfinished==false）时可以控制消费者进行多次消费。  
**（3）在main主程序中**，先对struct对象MyData的数据赋值，然后创建两个生产者与消费者线程，等待两个线程都执行完毕后关闭进程。为了验证多线程的实验效果，我在另一个cpp的main函数中创建了100个生产者线程和80个消费者线程。

# 实验步骤

1. **定义数据结构**

struct MyData{

HANDLE m\_S\_Empty;//生产者Semaphore

HANDLE m\_S\_Full;//消费者Semaphore

HANDLE m\_M\_Mutex;//互斥信号量

queue<int> food;//共享缓冲区队列

bool producerfinished;//标志着生产者是否结束生产

};

1. **生产者函数**

/\*生产者\*/

DWORD WINAPI Producer(void\* lp){

MyData \* md = (MyData\*)lp;

for(int i =0 ; i < 100; i++){

//测试缓冲区是否有空间

WaitForSingleObject(md->m\_S\_Empty, INFINITE);

/\*互斥访问缓冲区\*/

WaitForSingleObject(md->m\_M\_Mutex, INFINITE);

md->food.push(1); //放入缓冲区

printf("%d\t生产者生产1个产品，缓冲区共有%d个产品\t线程ID = %d\n",j++,md->food.size(),GetCurrentThreadId());

ReleaseMutex(md->m\_M\_Mutex); //释放互斥信号量

//将指定信号对象(有产品的缓冲区)的计数加一

ReleaseSemaphore(md->m\_S\_Full, 1, NULL);

}

md->producerfinished=true; //生产者结束生产，控制消费者线程结束

return 0;

}

1. **消费者函数**

/\*消费者\*/

DWORD WINAPI Consumer(void\* lp){

MyData \* md = (MyData\*)lp;

//若生产者没有结束生产

while(!md->producerfinished){

//测试缓冲区是否有产品

WaitForSingleObject(md->m\_S\_Full,INFINITE);

/\*互斥访问缓冲区\*/

WaitForSingleObject(md->m\_M\_Mutex,INFINITE);

md->food.pop(); //消费一个产品

printf("%d\t消费者消费1个产品，缓冲区还剩%d个产品\t线程ID = %d\n",j++,md->food.size(),GetCurrentThreadId());

ReleaseMutex(md->m\_M\_Mutex); //释放互斥信号量

//将指定信号对象(空的缓冲区)的计数加一

ReleaseSemaphore(md->m\_S\_Empty,1,NULL);

}

return 0;

}

1)其中涉及到ReleaseMutex函数，该函数释放指定互斥对象。如果函数调用成功，则返回值是非零值；如果函数调用失败，则返回值为0。其函数原型如下：

ReleaseMutex(HANDLE hMutex)；

参数hMutex：互斥对象句柄。为CreateMutex或OpenMutex函数的返回值。

2)涉及到WaitForSingleObject函数，该函数测试指定对象是否有信号或超时。如果函数调用成功，返回值表明引起函数返回的事件；如果函数调用失败，返回值是WAIT\_FAILED。其函数原型如下：

DWORD WaitForSingleObject(

HANDLE hHandle，DWORD dwMilliseconds)；

参数：hHandle：等待对象句柄。

dwMilliseconds：指定以毫秒为单位的超时间隔。

3)涉及到ReleaseSemaphore函数，该函数将指定信号对象的计数增加一个指定的数量。如果成功，则传回TRUE。否则传回FALSE。其函数原型如下：

ReleaseSemaphore(

HANDLE hSemaphore,

LONG lReleaseCount,

LPLONG lpPreviousCount);

1. **主程序（创建一个生产者和一个消费者线程）**

int main(){

/\*对各个信号量赋值\*/

MyData mydata; //创建一个MyData数据类型的实体mydata

mydata.m\_M\_Mutex = CreateMutex(NULL,false, NULL); //创建无名的互斥对象

/\*创建一个无名信号对象，初始计数为N，

如果成功就传回一个handle，否则传回NULL\*/

mydata.m\_S\_Empty = CreateSemaphore(NULL, N, N, NULL);

mydata.m\_S\_Full = CreateSemaphore(NULL, 0, N, NULL); //初始计数为0

mydata.producerfinished=false;//生产者结束标志刚开始设置为false,表示没有结束

//创建生产者和消费者线程

HANDLE handles[2];

handles[0] = CreateThread(NULL,0,&Producer,(void\*)&mydata,0,0);

handles[1] = CreateThread(NULL,0,&Consumer,(void\*)&mydata,0,0);

//等待两个线程都结束才往下执行

WaitForMultipleObjects(2, handles, true, INFINITE);

CloseHandle(mydata.m\_M\_Mutex);

CloseHandle(mydata.m\_S\_Full);

CloseHandle(mydata.m\_S\_Empty);

return 0;

}

1. 其中涉及到CreateThread线程创建函数，若函数调用成功，返回值为新线程的句柄；若函数调用失败,返回值为NULL。其函数原型如下：

HANDL CreateThread (

LPSECURITY\_ATTRIBUTES lpThreadAttributes, //其值决定返回的句柄是否可被子进程继承。若为NULL，则句柄不能被继承。

DWORD dwStackSize, //定义原始堆栈提交时的大小(按字节计)

LPTHREAD\_START\_ROUTINE lpStartAddress, //该指针表示远程进程中线程的起始地址

LPVOID lpParameter， //定义一个传递给该进程的32位值

DWORD dwCreationFlags, //定义控制进程创建的附加标志。若定义了CREATE\_SUSPENDED标志，线程创建时处于挂起状态，并且直到ResumeThread函数调用时才能运行。若该值为0，则该线程在创建后立即执行

LPDWORD lpThreadld); //指向一个32位值，它接受该线程的标识符

1. 也涉及到CreateMutex创建互斥对象函数。其函数原型如下：

HANDLE CreateMutex(

LPSECURITY\_ATTRIBUTES lpMutexAttributes，

BOOL bInitialOwner,

LPCTSTR lpName )；

3）还涉及到CreateSemapore函数，创建一个有名或者无名信号对象。如果成功就传回一个handle，否则传回NULL。其函数原型如下：

HANDLE CreateSemaphore(

LPSECURITY\_ATTRIBUTES lpAttributes,

LONG lInitialCount,

LONG lMaximumCount,

LPCTSTR lpName);

**（5）主程序（创建100个生产者和80个消费者线程）**

int main(){

/\*对各个信号量赋值\*/

MyData mydata; //创建一个MyData数据类型的实体mydata

mydata.m\_M\_Mutex = CreateMutex(NULL,false, NULL); //创建无名的互斥对象

/\*创建一个无名信号对象，初始计数为N，

如果成功就传回一个handle，否则传回NULL\*/

mydata.m\_S\_Empty = CreateSemaphore(NULL, N, N, NULL);

mydata.m\_S\_Full = CreateSemaphore(NULL, 0, N, NULL); //初始计数为0

mydata.producerfinished=false;//生产者结束标志刚开始设置为false,表示没有结束

//创建生产者和消费者线程

HANDLE procedure\_handles[100];

HANDLE customer\_handles[80];

for(int i=0;i<100;i++)

procedure\_handles[i] = CreateThread(NULL,0,&Producer,(void\*)&mydata,0,0);

for(int j=0;j<80;j++)

customer\_handles[j] = CreateThread(NULL,0,&Consumer,(void\*)&mydata,0,0);

//等待所有线程都结束才往下执行

WaitForMultipleObjects(2, procedure\_handles, true, INFINITE);

WaitForMultipleObjects(2, customer\_handles, true, INFINITE);

CloseHandle(mydata.m\_M\_Mutex);

CloseHandle(mydata.m\_S\_Full);

CloseHandle(mydata.m\_S\_Empty);

return 0;

}

# 实验结果

1. **创建一个生产者和一个消费者线程，实现进程的互斥与同步。**

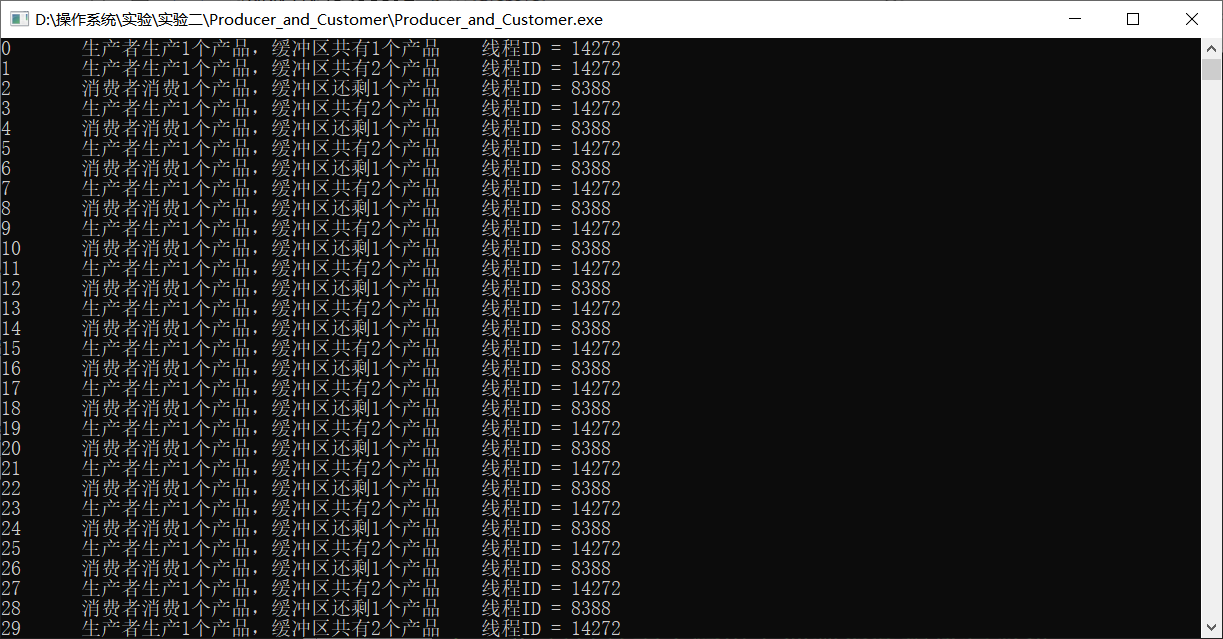


图5-1-1 1个生产者和1个消费者线程执行结果

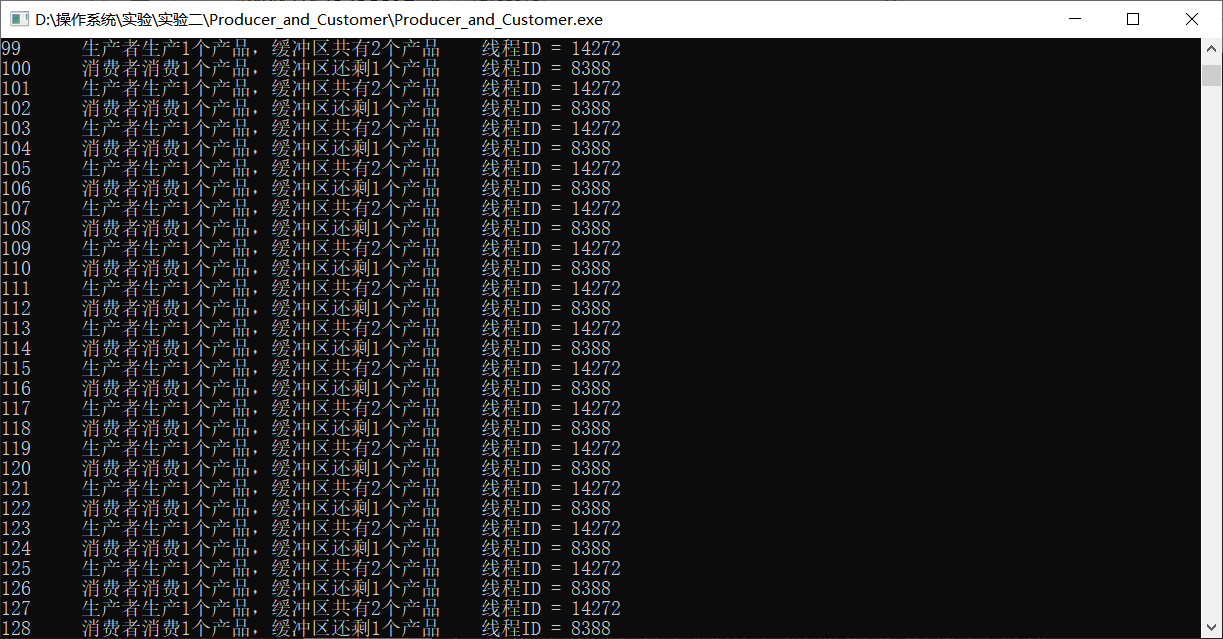


图5-1-2 1个生产者和1个消费者线程执行结果

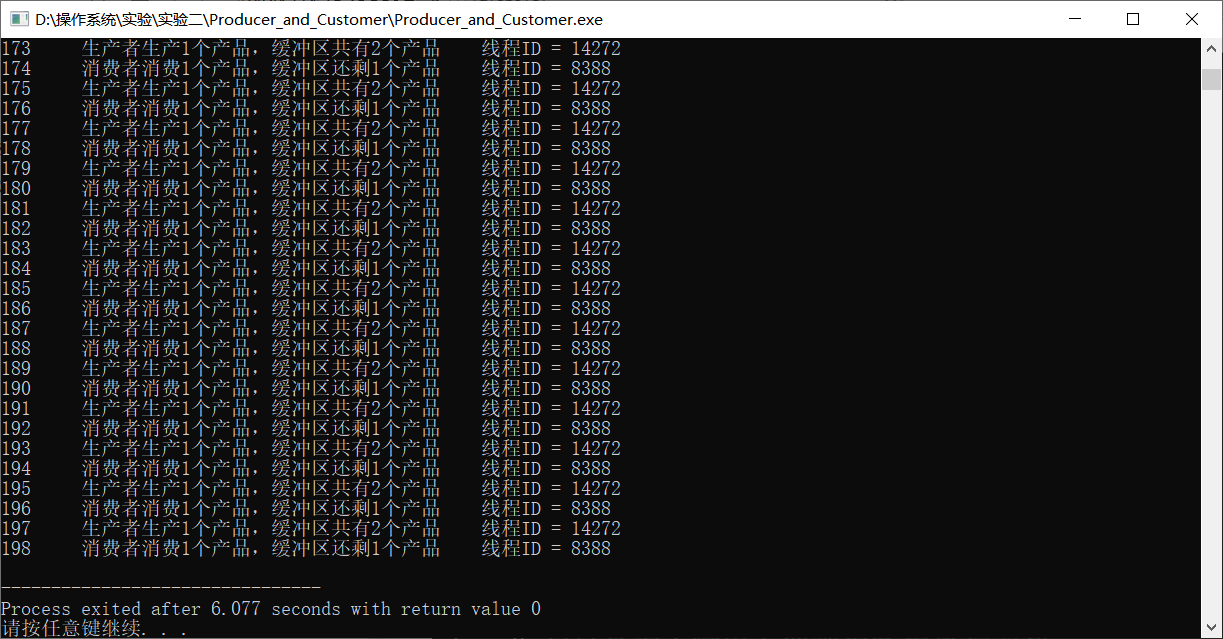


图5-1-3 1个生产者和1个消费者线程执行结果

1. **创建100个生产者和80个消费者线程，实现进程的互斥与同步。**

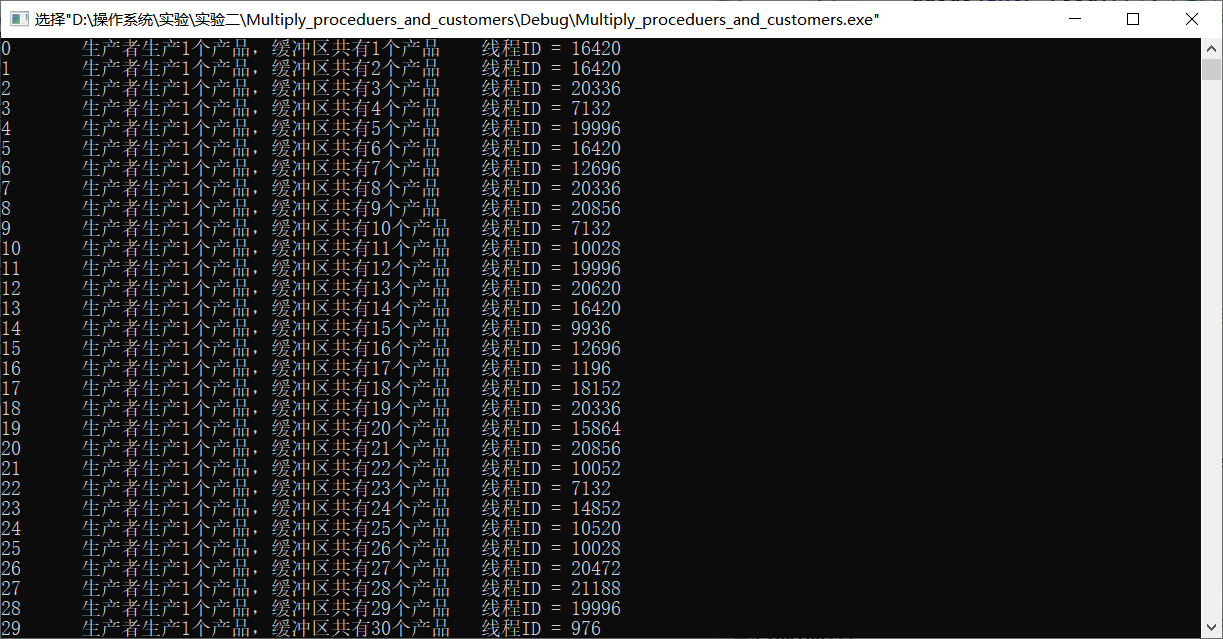


图5-2-1 100个生产者和80个消费者线程执行结果

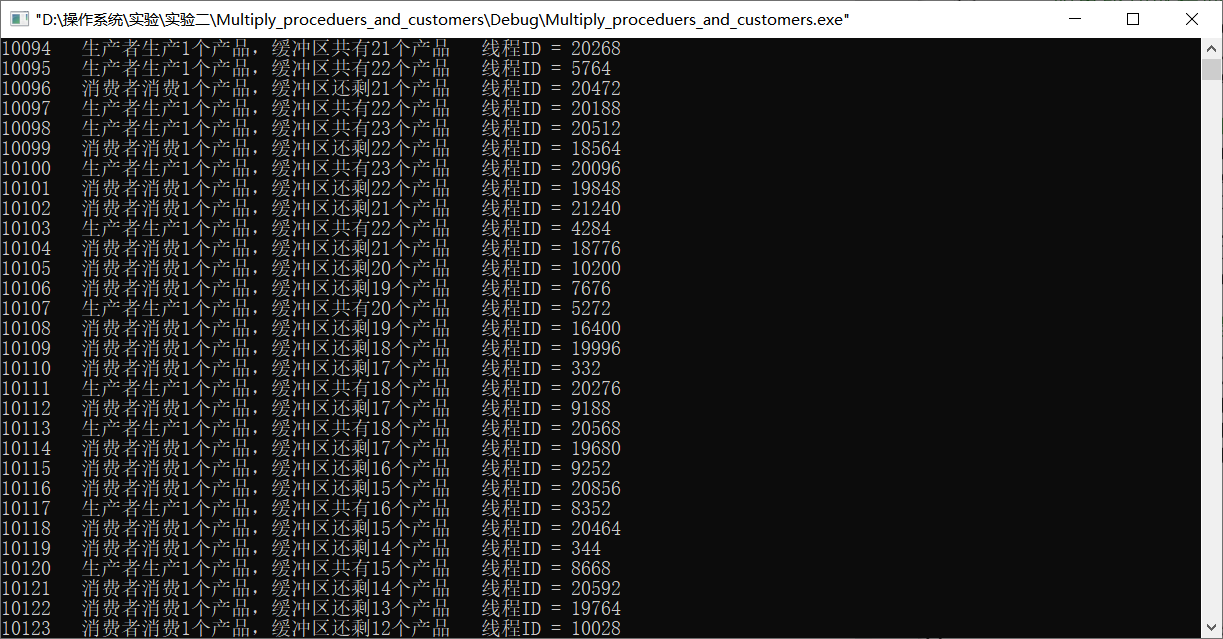


图5-2-2 100个生产者和80个消费者线程执行结果

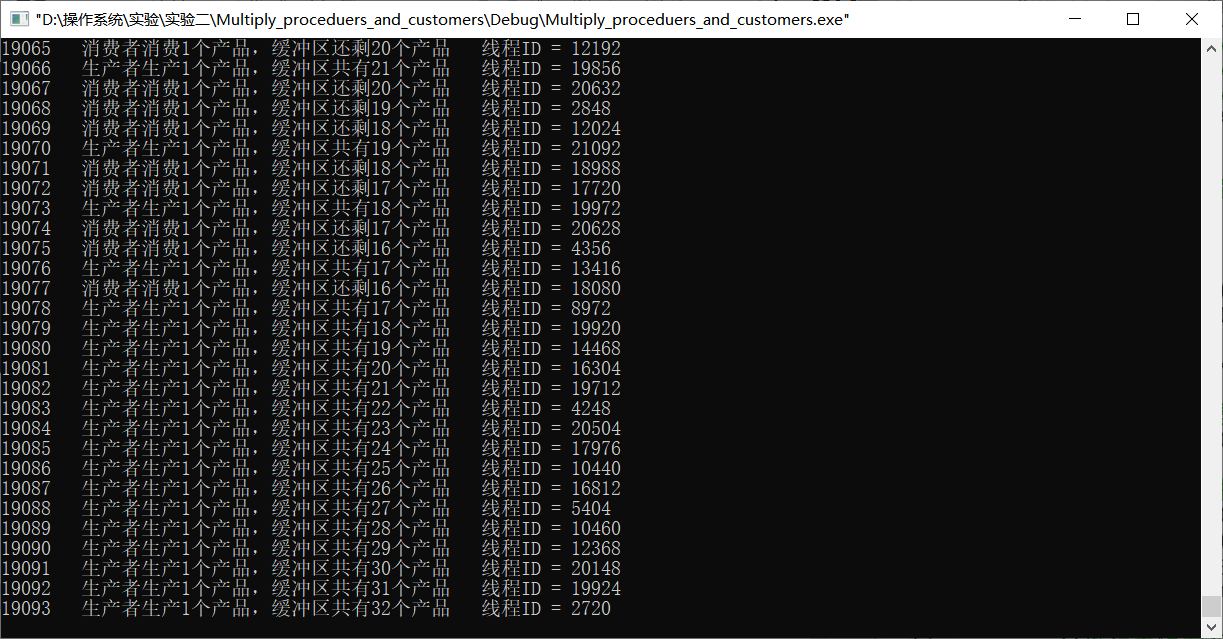


图5-2-3 100个生产者和80个消费者线程执行结果

# 实验中遇到的问题及解决方法

以下是我在写程序时遇到的一些小问题：  
（1）我一开始没有创建struct，但我发现许多类似编程都会创建一个结构体，把线程中要用到的数据信号量都放在结构体中。我经过思考发现在main函数中创建线程的时候可以直接通过参数(void\*)&mydata将所有线程要使用的数据都传入线程，这样创建线程比较方便。  
（2）考虑到如果把生产者和消费者函数体都放在while(true)循环中会使程序一直运行，无法停止，为了避免这种情况，我在struct中增加了bool类型的producerfinished变量，通过producerfinished变量控制while循环次数，当生产者生产完100个产品后，它的值变为true,使得消费者进程会退出while循环，不能继续消费。  
（3）我原先在CreateThread创建了生产者和消费者线程后直接CloseHandle，运行后发现程序没有输出，通过思考我认为是main函数没有等待线程的执行就结束了，为了避免这种情况，我在CloseHandle之前增加了WaitForMultipleObjects操作，也就是要等到2个线程都结束以后才能继续往下执行。它的第三个参数若为false，则只要其中一个线程结束了就可以继续往下执行。

# 实验体会

1. 在做本次生产者与消费者进程之间互斥和同步的实验时，我结合了老师提供的ppt上的程序伪代码以及操作系统课程中并发章节的内容，通过自己的思考理解重新整理编写代码，验证线程的同步和互斥操作的正确性，使我对进程的并发主题有了更深层次的理解。
2. **信号量。**信号量是特殊的变量，用于发送信号。

一个进程为了通过信号量s发送信号,它需要执行原语 semSignal(s)/V(s)，使信号量值增1。如果值小于或等于零，表示之前有进程在等该信号，则需要在该信号量的阻塞队列中唤醒一个进程。

一个进程通过信号量s接收信号, 它需要执行原语semWait(s) /P(s)，使信号量值减1，如果信号量值变为负数，则执行该操作的进程被阻塞。如果相应的信号没有接收到，该进程将被挂起，直接它所需的信号发送为止。

其中，S>0表示有S个资源可用；S=0表示无资源可用；S<0则| S |表示S等待队列中的进程个数。注意：信号量的初值应该大于等于0。

（3）**互斥信号量。**互斥信号量是为一组需要互斥访问共享临界资源的并发进程设置的，每个进程均可对它施加wait、signal操作，即可申请和释放该临界资源，其初始值置为1。互斥操作时，wait(P)/signal(V)操作处于同一进程，若wait(S1)和wait(S2)两个操作在一起，那么wait操作的顺序至关重要,一个同步wait操作与一个互斥wait操作在一起时同步wait操作在互斥wait操作前，否则会发生死锁。

（4）**同步信号量。**同步信号量要保证多个进程执行顺序的正确性。两个或多个进程可以通过简单的信号进行合作，一个进程可以被迫在某一位置停止，直到它接收到特定的信号。只有拥有该资源的进程才能对同步信号量进行wait操作（即申请资源），而其合作进程只能对它进行signal操作（即释放资源）。也就是说，先做动作的进程C在动作完成。后对同步信号量施加signal操作，发送信号；后做动作的进程P在动作前对同步信号量施加wait操作，测试消息是否到达。同步操作时，wait(P)/signal(V)不在同一进程中出现。

（5）综上，考虑一个具体的问题时，我们应当先确定问题中有几个同步关系、几个互斥关系，并使用相应的信号量实现并发操作。