[秒杀多线程第十篇 生产者消费者问题](http://blog.csdn.net/morewindows/article/details/7577591)

分类： [Windows多线程](http://blog.csdn.net/MoreWindows/article/category/1115271) [Windows编程](http://blog.csdn.net/MoreWindows/article/category/862060)2012-05-21 10:18 61043人阅读 [评论](http://blog.csdn.net/morewindows/article/details/7577591#comments)(114) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/morewindows/article/details/7577591#report)

[多线程](http://www.csdn.net/tag/%e5%a4%9a%e7%ba%bf%e7%a8%8b)[null](http://www.csdn.net/tag/null)[buffer](http://www.csdn.net/tag/buffer)[产品](http://www.csdn.net/tag/%e4%ba%a7%e5%93%81)[semaphore](http://www.csdn.net/tag/semaphore)[output](http://www.csdn.net/tag/output)

    继[经典线程同步问题](http://blog.csdn.net/morewindows/article/details/7442333)之后，我们来看看生产者消费者问题及读者写者问题。生产者消费者问题是一个著名的线程同步问题，该问题描述如下：有一个生产者在生产产品，这些产品将提供给若干个消费者去消费，为了使生产者和消费者能并发执行，在两者之间设置一个具有多个缓冲区的缓冲池，生产者将它生产的产品放入一个缓冲区中，消费者可以从缓冲区中取走产品进行消费，显然生产者和消费者之间必须保持同步，即不允许消费者到一个空的缓冲区中取产品，也不允许生产者向一个已经放入产品的缓冲区中再次投放产品。

    这个生产者消费者题目不仅常用于操作系统的课程设计，也常常在程序员和软件设计师考试中出现。并且在计算机考研的专业课考试中也是一个非常热门的问题。因此现在就针对这个问题进行详细深入的解答。

    首先来简化问题，先假设生产者和消费者都只有一个，且缓冲区也只有一个。这样情况就简便多了。

    第一．从缓冲区取出产品和向缓冲区投放产品必须是互斥进行的。可以用[关键段](http://blog.csdn.net/morewindows/article/details/7442639)和[互斥量](http://blog.csdn.net/morewindows/article/details/7470936)来完成。

    第二．生产者要等待缓冲区为空，这样才可以投放产品，消费者要等待缓冲区不为空，这样才可以取出产品进行消费。并且由于有二个等待过程，所以要用二个[事件](http://blog.csdn.net/morewindows/article/details/7445233)或[信号量](http://blog.csdn.net/morewindows/article/details/7481609)来控制。（有两个同步过程所以需要两个信号量（或事件）来控制）。我对事件和信号量的理解：事件和信号量都是内核对象，它们都能用于同步和互斥，但是事件只能用于对一个资源的同步，而信号量可以用于对多个事件的同步（如在生产者-消费者问题中当只有一个缓冲区时可以用事件来实现同步也可以用信号量来实现两步，但是当有多个缓冲区时只能用信号量来实现同步。原因在于信号量大于0的情况均表示触发，而事件的1表示触发）。

    考虑这二点后，代码很容易写出来。另外为了美观起见，将消费者的输出颜色设置为彩色，有关如何在控制台下设置彩色输出请参阅《[VC 控制台颜色设置](http://blog.csdn.net/morewindows/article/details/6789206)》。

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7577591)

1. //1生产者 1消费者 1缓冲区
2. //使用二个事件，一个表示缓冲区空，一个表示缓冲区满。
3. //再使用一个关键段来控制缓冲区的访问
4. #include <stdio.h>
5. #include <process.h>
6. #include <windows.h>
7. //设置控制台输出颜色
8. **BOOL** SetConsoleColor(**WORD** wAttributes)
9. {
10. **HANDLE** hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
11. **if** (hConsole == INVALID\_HANDLE\_VALUE)
12. **return** FALSE;
13. **return** SetConsoleTextAttribute(hConsole, wAttributes);
14. }
15. **const** **int** END\_PRODUCE\_NUMBER = 10;   //生产产品个数
16. **int** g\_Buffer;                        //缓冲区
17. //事件与关键段
18. CRITICAL\_SECTION g\_cs;
19. **HANDLE** g\_hEventBufferEmpty, g\_hEventBufferFull;
20. //生产者线程函数
21. unsigned **int** \_\_stdcall ProducerThreadFun(**PVOID** pM)
22. {
23. **for** (**int** i = 1; i <= END\_PRODUCE\_NUMBER; i++)
24. {
25. //等待缓冲区为空
26. WaitForSingleObject(g\_hEventBufferEmpty, INFINITE);
28. //互斥的访问缓冲区
29. EnterCriticalSection(&g\_cs);
30. g\_Buffer = i;
31. printf("生产者将数据%d放入缓冲区\n", i);
32. LeaveCriticalSection(&g\_cs);
34. //通知缓冲区有新数据了
35. SetEvent(g\_hEventBufferFull);
36. }
37. **return** 0;
38. }
39. //消费者线程函数
40. unsigned **int** \_\_stdcall ConsumerThreadFun(**PVOID** pM)
41. {
42. **volatile** **bool** flag = **true**;
43. **while** (flag)
44. {
45. //等待缓冲区中有数据
46. WaitForSingleObject(g\_hEventBufferFull, INFINITE);
48. //互斥的访问缓冲区
49. EnterCriticalSection(&g\_cs);
50. SetConsoleColor(FOREGROUND\_GREEN);
51. printf("  消费者从缓冲区中取数据%d\n", g\_Buffer);
52. SetConsoleColor(FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);
53. **if** (g\_Buffer == END\_PRODUCE\_NUMBER)
54. flag = **false**;
55. LeaveCriticalSection(&g\_cs);
57. //通知缓冲区已为空
58. SetEvent(g\_hEventBufferEmpty);
60. Sleep(10); //some other work should to do
61. }
62. **return** 0;
63. }
64. **int** main()
65. {
66. printf("  生产者消费者问题   1生产者 1消费者 1缓冲区\n");
67. printf(" -- by MoreWindows( http://blog.csdn.net/MoreWindows ) --\n\n");
69. InitializeCriticalSection(&g\_cs);
70. //创建二个自动复位事件，一个表示缓冲区是否为空，另一个表示缓冲区是否已经处理
71. g\_hEventBufferEmpty = CreateEvent(NULL, FALSE, TRUE, NULL);
72. g\_hEventBufferFull = CreateEvent(NULL, FALSE, FALSE, NULL);
74. **const** **int** THREADNUM = 2;
75. **HANDLE** hThread[THREADNUM];
77. hThread[0] = (**HANDLE**)\_beginthreadex(NULL, 0, ProducerThreadFun, NULL, 0, NULL);
78. hThread[1] = (**HANDLE**)\_beginthreadex(NULL, 0, ConsumerThreadFun, NULL, 0, NULL);
79. WaitForMultipleObjects(THREADNUM, hThread, TRUE, INFINITE);
80. CloseHandle(hThread[0]);
81. CloseHandle(hThread[1]);
83. //销毁事件和关键段
84. CloseHandle(g\_hEventBufferEmpty);
85. CloseHandle(g\_hEventBufferFull);
86. DeleteCriticalSection(&g\_cs);
87. **return** 0;
88. }

运行结果如下所示：



可以看出生产者与消费者已经是有序的工作了。

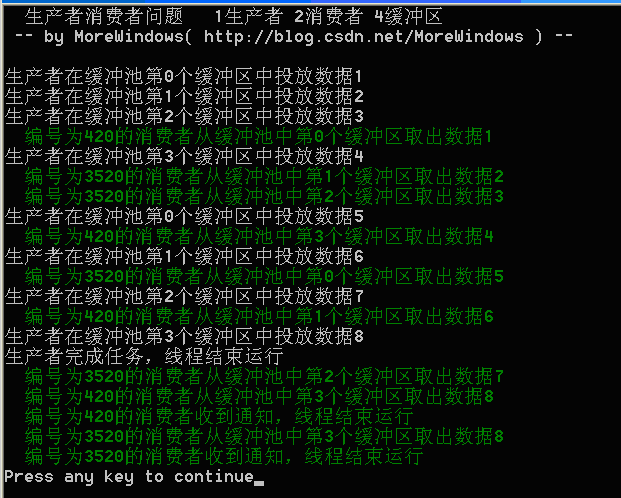
然后再对这个简单生产者消费者问题加大难度。将消费者改成2个，缓冲池改成拥有4个缓冲区的大缓冲池

    如何来思考了这个问题了？首先根据上面分析的二点，可以知道生产者和消费者由一个变成多个的影响不大，唯一要注意的是缓冲池变大了，回顾一下《[秒杀多线程第八篇 经典线程同步 信号量Semaphore](http://blog.csdn.net/morewindows/article/details/7481609)》中的信号量，不难得出用二个信号量就可以解决这种缓冲池有多个缓冲区的情况——用一个信号量A来记录为空的缓冲区个数，另一个信号量B记录非空的缓冲区个数，然后生产者等待信号量A，消费者等待信号量B就可以了。因此可以仿照上面的代码来实现复杂生产者消费者问题，示例代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7577591)

1. //1生产者 2消费者 4缓冲区
2. #include <stdio.h>
3. #include <process.h>
4. #include <windows.h>
5. //设置控制台输出颜色
6. **BOOL** SetConsoleColor(**WORD** wAttributes)
7. {
8. **HANDLE** hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
9. **if** (hConsole == INVALID\_HANDLE\_VALUE)
10. **return** FALSE;
12. **return** SetConsoleTextAttribute(hConsole, wAttributes);
13. }
14. **const** **int** END\_PRODUCE\_NUMBER = 8;  //生产产品个数
15. **const** **int** BUFFER\_SIZE = 4;          //缓冲区个数
16. **int** g\_Buffer[BUFFER\_SIZE];          //缓冲池
17. **int** g\_i, g\_j;
18. //信号量与关键段
19. CRITICAL\_SECTION g\_cs;
20. **HANDLE** g\_hSemaphoreBufferEmpty, g\_hSemaphoreBufferFull;
21. //生产者线程函数
22. unsigned **int** \_\_stdcall ProducerThreadFun(**PVOID** pM)
23. {
24. **for** (**int** i = 1; i <= END\_PRODUCE\_NUMBER; i++)
25. {   //等待有空的缓冲区出现
26. WaitForSingleObject(g\_hSemaphoreBufferEmpty, INFINITE);
28. //互斥的访问缓冲区
29. EnterCriticalSection(&g\_cs);
30. g\_Buffer[g\_i] = i;
31. printf("生产者在缓冲池第%d个缓冲区中投放数据%d\n", g\_i, g\_Buffer[g\_i]);
32. /\* g\_i在0~ BUFFER\_SIZE-1之间循环计数。 g\_i = (g\_i + 1) % BUFFER\_SIZE🡸效果上等价于🡺 g\_i = (g\_i + 1); g\_i = g\_i% BUFFER\_SIZE。 (g\_i + 1)使计数值能循环，% BUFFER\_SIZE使得取值在0~ BUFFER\_SIZE-1之间。这使得g\_Buffer[g\_i]像是一个循环表 \*/
33. g\_i = (g\_i + 1) % BUFFER\_SIZE;
34. LeaveCriticalSection(&g\_cs);
36. //通知消费者有新数据了
37. ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);
38. }
39. printf("生产者完成任务，线程结束运行\n");
40. **return** 0;
41. }
42. //消费者线程函数
43. unsigned **int** \_\_stdcall ConsumerThreadFun(**PVOID** pM)
44. {
45. **while** (**true**)
46. {
47. //等待非空的缓冲区出现
48. WaitForSingleObject(g\_hSemaphoreBufferFull, INFINITE);
50. //互斥的访问缓冲区
51. EnterCriticalSection(&g\_cs);
52. SetConsoleColor(FOREGROUND\_GREEN);
53. printf("  编号为%d的消费者从缓冲池中第%d个缓冲区取出数据%d\n", GetCurrentThreadId(), g\_j, g\_Buffer[g\_j]);
54. SetConsoleColor(FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);
55. **if** (g\_Buffer[g\_j] == END\_PRODUCE\_NUMBER)//结束标志
56. {
57. LeaveCriticalSection(&g\_cs);
58. //通知其它消费者有新数据了(结束标志)。防止另一个线程一直在等资源(此时生产者生产的产品已经消费完了，但另一个线程可能还在等资源，所以这里做一个假生产动作)。但是这样会出现一个问题，就是导致最后一个产品会被消费两次。解决方法是设置一个全局结束标志。详见评论
59. ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);
60. **break**;
61. }
62. g\_j = (g\_j + 1) % BUFFER\_SIZE; /\* g\_j在0~ BUFFER\_SIZE-1之间循环计数。 (g\_j + 1)与赋值使计数值增加，% BUFFER\_SIZE使得取值在0~ BUFFER\_SIZE-1之间。这使得g\_Buffer[g\_j]像是一个循环表 \*/
63. LeaveCriticalSection(&g\_cs);
65. Sleep(50); //some other work to do
67. ReleaseSemaphore(g\_hSemaphoreBufferEmpty, 1, NULL);
68. }
69. SetConsoleColor(FOREGROUND\_GREEN);
70. printf("  编号为%d的消费者收到通知，线程结束运行\n", GetCurrentThreadId());
71. SetConsoleColor(FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);
72. **return** 0;
73. }
74. **int** main()
75. {
76. printf("  生产者消费者问题   1生产者 2消费者 4缓冲区\n");
77. printf(" -- by MoreWindows( http://blog.csdn.net/MoreWindows ) --\n\n");
79. InitializeCriticalSection(&g\_cs);
80. //初始化信号量,一个记录有产品的缓冲区个数,另一个记录空缓冲区个数.
81. g\_hSemaphoreBufferEmpty = CreateSemaphore(NULL, 4, 4, NULL);
82. g\_hSemaphoreBufferFull  = CreateSemaphore(NULL, 0, 4, NULL);
83. g\_i = 0;
84. g\_j = 0;
85. memset(g\_Buffer, 0, **sizeof**(g\_Buffer));
87. **const** **int** THREADNUM = 3;
88. **HANDLE** hThread[THREADNUM];
89. //生产者线程
90. hThread[0] = (**HANDLE**)\_beginthreadex(NULL, 0, ProducerThreadFun, NULL, 0, NULL);
91. //消费者线程
92. hThread[1] = (**HANDLE**)\_beginthreadex(NULL, 0, ConsumerThreadFun, NULL, 0, NULL);
93. hThread[2] = (**HANDLE**)\_beginthreadex(NULL, 0, ConsumerThreadFun, NULL, 0, NULL);
94. WaitForMultipleObjects(THREADNUM, hThread, TRUE, INFINITE);
95. **for** (**int** i = 0; i < THREADNUM; i++)
96. CloseHandle(hThread[i]);
98. //销毁信号量和关键段
99. CloseHandle(g\_hSemaphoreBufferEmpty);
100. CloseHandle(g\_hSemaphoreBufferFull);
101. DeleteCriticalSection(&g\_cs);
102. **return** 0;
103. }

运行结果如下图所示：



输出结果证明各线程的同步和互斥已经完成了。

------我的疑问是：消费者从缓冲区中取数据应该是随意的（也就是说只要缓冲区是有数据的，消费者就可以从中取，而不需要按照什么顺序）。我们不应该假设消费者从缓冲区中取数据的顺序。

----解答：在有些情况下是需要确定消费者从缓冲区中最数据的顺序的。如：播放器中，生产者生产一帧一帧的画面数据放入缓冲区中，消费者按照生产者生产的顺序来取数据显示，如果不按照生产的顺序来取数据那么显示的画面就会是混乱的。

至此，生产者消费者问题已经圆满的解决了，下面作个总结：

1．首先要考虑生产者与消费者对缓冲区操作时的互斥。

2．不管生产者与消费者有多少个，缓冲池有多少个缓冲区。都只有二个同步过程——分别是生产者要**等待**有空缓冲区才能投放产品，消费者要**等待**有非空缓冲区才能去取产品。

下一篇《秒杀多线程第十一篇读者写者问题》将介绍另一个著名的同步问题——读者写者问题，欢迎大家再来参阅。

[](http://blog.csdn.net/woriwoshi)

楼主，你第二个例子（//1生产者 2消费者 4缓冲区 ）  
最后显示数字8 被消费了两次？？？？

回复woriwoshi：为了让另外一个消费者结束线程而已，你也可以用别的方法，比如设置一个over变量，第一个读到8的消费者把over置为1，然后释放一个信号量，另一个消费者先检测over，如果over为1，则结束线程即可。

[](http://blog.csdn.net/mashibing1990)

**[cpp]** [view plaincopy](http://blog.csdn.net/morewindows/article/details/7577591)

1. //消费者线程函数中while函数改写
2. **while** (**true**)
3. {
4. //等待非空的缓冲区出现
5. WaitForSingleObject(g\_hSemaphoreBufferFull, INFINITE);
7. //互斥的访问缓冲区
8. EnterCriticalSection(&g\_cs);
10. **if** (g\_bProduceOver == **true**)
11. {
12. **break**;
13. }
14. SetConsoleColor(FOREGROUND\_GREEN);
15. printf("  编号为%d的消费者从缓冲池中第%d个缓冲区取出数据%d\n", GetCurrentThreadId(), g\_j, g\_Buffer[g\_j]);
16. SetConsoleColor(FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGRO11111UND\_BLUE);
17. **if** (g\_Buffer[g\_j] == END\_PRODUCE\_NUMBER)//结束标志
18. {
19. //通知其它消费者有新数据了(结束标志)
20. ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);
21. Sleep(100);
22. g\_bProduceOver = **true**;
23. LeaveCriticalSection(&g\_cs);
24. **break**;
25. }
26. g\_j = (g\_j + 1) % BUFFER\_SIZE;
27. LeaveCriticalSection(&g\_cs);
28. Sleep(50); //some other work to do
30. ReleaseSemaphore(g\_hSemaphoreBufferEmpty, 1, NULL);
31. }

加入一个表征生产者结束生产的标记。并且将判断放在关键段中的开始部分。这样则不会出现取9次（数据8被取两次）的情况。

49楼 [TuTuJiangHuaSheng](http://blog.csdn.net/TuTuJiangHuaSheng) 2014-06-09 11:12发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7577591#reply)

[](http://blog.csdn.net/TuTuJiangHuaSheng)

为什么生产者生产了8次，而消费者却消费了9次?

回复TuTuJiangHuaSheng：你可以把下面这段注释掉，然后看看效果，这里的本意是为了结束线程，但造成了消费重复，你可以自定义结束标志来结束线程  
if (g\_Buffer[g\_j] == END\_PRODUCE\_NUMBER)//结束标志   
{   
LeaveCriticalSection(&g\_cs);   
//通知其它消费者有新数据了(结束标志)   
ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);   
break;   
}

[http://avatar.csdn.net/5/6/D/3_saiksy.jpg](http://blog.csdn.net/SaikSy)

第一个例子没必要加临界区，而且加了也是无效的，临界区是用在多条线程共运行同一段代码，此处2条线程各自运行自己的代码块，互不相干，同步的话，用事件就行。（这种说法有问题。但在这个例子中要不要加临界区确是一个问题）

回复SaikSy：我也发现，第一个缓冲区控制是多余的，生产者和消费者进程是先后进行的，先后访问缓冲区的，不会产生互斥访问的问题。

回复xlf13872135090：我也这么认为

回复u011327682：第一个是缓冲区是单个，所以不用不会出现同时访问的情况；当缓冲区为多个的时候，则有可能出现这种情况，楼主这种情况更利于程序扩展。

[](http://blog.csdn.net/springontime)

感谢楼主，有不少收获，但是不知道为什么，总感觉第二个例子有问题，其中之一就是，消费者取数据时候被互斥了，导致一个消费者取得的时候别的另一个消费者等着，和第一个例子相比，就好像一个人做的事情交给两给人轮流去干了，就好比要将一颗钉子钉入墙里面，本来一个人用锤子敲打就行了，现在变成两个人轮流捶打，有点多余（计算机不怕累），且速度还会降低。另外一个问题就是虽然设置了4个缓冲区，但嫣然放是按缓冲区序号由小到大放数据，取是按序号由小到大取数据，这完全发挥不出多缓冲区的效果。

回复springontime：呵呵，这是比较基础的示范，还有很大的改进空间。

[](http://blog.csdn.net/xhmstory)

你好，这个可以用在实时的视频播放上面吧，就是消费者一直在消费，生产者等缓冲区空了马上生产。

35楼 [yingxiong19](http://blog.csdn.net/yingxiong19) 2013-09-04 10:22发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7577591#reply)

[](http://blog.csdn.net/yingxiong19)

您好，请教您一个问题，按照您上面的实现，多生产者多消费者多缓冲区的情况下，生产者和消费者使用的是同一把锁EnterCriticalSection(&g\_cs); 这里我有点不明白，我觉得在多缓冲区的情况下，生产者和消费者是可以并行的，需要互斥的应该是生产者与生产者之间、消费者与消费者之间，所以应该使用不同的锁

回复yingxiong19：用EnterCriticalSection( &g\_cs );有一个原因是，在消费者线程里面设置控制台颜色，并输出字符，如果不和生产者互斥，那么会造成生产者的部分显示也是消费者的颜色，或者字体错乱。

[](http://blog.csdn.net/bigfox2007750219)

我觉得第一个问题生产者和消费者对缓冲区没必要互斥进行。。。。(关于进程互斥可以看操作系统原理视频5-2)  
（1）一个生产者，一个消费者，公用一个缓冲区。  
定义两个同步信号量：  
empty——表示缓冲区是否为空，初值为1。  
full——表示缓冲区中是否为满，初值为0。  
生产者进程  
while(TRUE){  
生产一个产品;  
P(empty);  
产品送往Buffer;  
V(full);  
}  
消费者进程  
while(True){  
P(full);  
从Buffer取出一个产品;  
V(empty);  
消费该产品;  
}

[http://avatar.csdn.net/D/7/D/3_a784063999.jpg](http://blog.csdn.net/a784063999)

一個緩衝區可以不用關鍵段嗎?

回复a784063999：如果生产者和消费者都只有一个，那可以不用，因此生产者和消费者已经做了同步了，且不存在多个生产者或消费者同时同时访问缓冲区的情况。

13楼 [hahasteve](http://blog.csdn.net/hahasteve) 2012-09-15 22:02发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7577591#reply)

[](http://blog.csdn.net/hahasteve)

楼主我想问为什么最后一次两个consumer线程对3号缓冲区各取了一次啊 ??? 3520去取的时候不是应该empty了吗?

回复hahasteve：此题的解法是有些漏洞，明显其中拿到最后的数据的线程为了不致让后来的线程阻塞，调用ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);   
通知后面的线程，但后面线程一段进入EnterCriticalSection(&g\_cs); 互斥段，不可避免的取出数据，导致错误。解决此题有两种方案  
1. 在线程设置bool变量，任一线程进入了  
if (g\_Buffer[g\_j] == END\_PRODUCE\_NUMBER)//结束标志   
设置其为true,表示数据完结，当其它的消费进程进入互斥段之后，如果为true,即退出，最终线程通知运行完毕。  
2. 另一个是设置g\_Buffer[g\_j]++，越界信号，这样其它消费线程做个比较判断，就可以顺利退出。  
下一步，我将在程序中来实现，修正bug.

Re: [MoreWindows](http://blog.csdn.net/MoreWindows) 2012-09-19 22:43发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7577591#reply)

[http://avatar.csdn.net/C/F/7/3_morewindows.jpg](http://blog.csdn.net/MoreWindows)

回复hahasteve：本文中用结束标志来控制消费者线程是否继续执行，当消费者线程读到结束标志后不会取出数据（修改全局的访问下标g\_j）而是直接退出。  
55. if (g\_Buffer[g\_j] == END\_PRODUCE\_NUMBER)//结束标志   
56. {   
57. LeaveCriticalSection(&g\_cs);   
58. //通知其它消费者有新数据了(结束标志)   
59. ReleaseSemaphore(g\_hSemaphoreBufferFull, 1, NULL);   
60. break;   
61. }

[](http://blog.csdn.net/wodownload2)

回复tigerxuchuanli：我觉得这个代码是可以的，不知道楼主可否帮我看下：  
unsigned int \_\_stdcall ConsumerThread(PVOID pM)  
{   
while(true)  
{  
  
WaitForSingleObject(semaphore\_full,INFINITE);  
EnterCriticalSection(&mutex);  
if(!isover)  
{  
char temp=buffer[g\_p\_pc];  
if(temp=='a'+PRODUCTNUM-1)  
{  
isover=true;  
printf("\t消费者 %d 拿到最后一个产品\n",GetCurrentThreadId(),temp);  
ReleaseSemaphore(semaphore\_full,1,NULL);  
LeaveCriticalSection(&mutex);  
break;  
}  
printf("\t消费者 %d 从缓冲区 %d 拿到产品 %c\n",GetCurrentThreadId(),g\_p\_pc,temp);  
g\_p\_pc=(g\_p\_pc+1)%BUFFERSIZE;  
ReleaseSemaphore(semaphore\_empty,1,NULL);  
LeaveCriticalSection(&mutex);  
}  
else  
{  
ReleaseSemaphore(semaphore\_full,1,NULL);  
LeaveCriticalSection(&mutex);  
break;  
}  
}  
printf("消费者 %d 线程结束\n",GetCurrentThreadId());  
return 0;  
}

回复tigerxuchuanli：弱弱的问下，作者的第2个例子，消费者代码，感觉有点问题，很显然其他线程也可以读取出最后一个产品。这个人（tigerxuchuanli）说的方法是对的。

[](http://blog.csdn.net/rwxdfbb)

可以查下《Windows内核情景分析》和《windows内核原理与实现》

6楼 [rwxdfbb](http://blog.csdn.net/rwxdfbb) 2012-05-31 15:31发表 [[回复]](http://blog.csdn.net/morewindows/article/details/7577591#reply)

[](http://blog.csdn.net/rwxdfbb)！！！！！！！！！确实是个问题，有待改进

大致看了一下，觉得楼主的第一个条件（**生产者与消费者对缓冲区操作的互斥）是不是多此一举，而且不符合实际需求。**根据两个信号量，生产者和消费者同时访问缓冲区是不会冲突的，因为生产者不会在消费者还没取走缓冲区的地址写入数据，消费者也不会在生产者没放的缓冲区地址取走数据，唯一需要互斥的可能是多个消费者之间的访问，他们有可能访问同一个缓冲区地址。  
其次，实际应用者的角度来看，生产者只管按照自己的节奏生产，只要有地方能存放生产出来的东西，不需要管消费者现在是不是在消费；消费者只管消费，只要有东西可消费，不需要知道现在生产者是不是在生产东西。

回复zhouzhou423：我的想法是对每一个缓冲池是否被生产者占用加一个互斥，互斥产生在生产者之间；缓冲池是否被消费者占用加一个互斥，互斥产生在消费者之间。这样应该就可以最大化的使每一个线程等待最少的时间。不知道我想的对不对。