**什么是生产者与消费者同步问题**

生产者消费者问题，也称有限缓冲问题，是一个多线程同步问题的经典案例。该问题描述了两个共享固定大小[缓冲区](http://baike.so.com/doc/2576817-2721104.html)的[线程](http://baike.so.com/doc/115404-121766.html)--即所谓的"生产者"和"消费者"--在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区中空时消耗数据。

知道什么是生产者和消费者问题，那么我们就开始程序设计。

首先我们进行最原始的设计，看看会产生什么问题？

#include <toyix.h>

int goods=0;

void producer()

{

while(1)

{

goods++;

delay(1000);

printf("\n productd!! the rooom have %d goods\n ",goods);

}

}

void consumer()

{

while(1)

{

goods--;

delay(2000);

printf("\n consumed!! the rooom have %d goods\n ",goods); }

}

main()

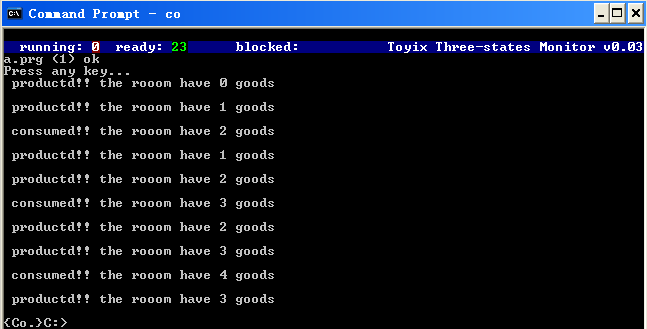
{

cobegin(producer, consumer,0);

getch();

}

我们编译连接后运行，得到结果如下：



分析这个结果，你可以看到发生了错误，比如第二行说生产了1个货物，第三行说消费了，但是货物怎么变成两个，你说明我们两个轻权进程没有同步。发生了混乱，也就是说生产者和消费者在共享这个资源的时候必须互斥。

#include <toyix.h>

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

p(&mutex);

goods++;

delay(1000);

printf("\n productd!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

void consumer()

{

while(1)

{

p(&mutex);

goods--;

delay(2000);

printf("\n consumed!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

main()

{

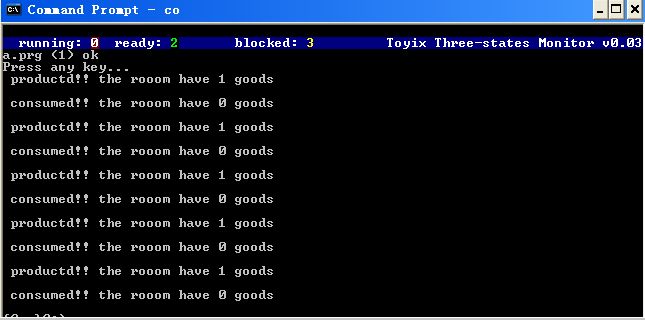
set(&mutex,1);

cobegin(producer, consumer,0);

getch();

}

得到的结果如下：

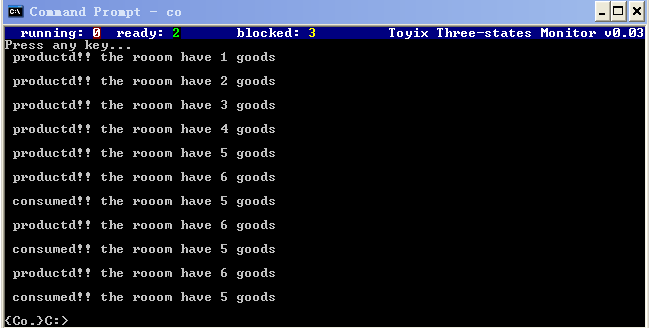


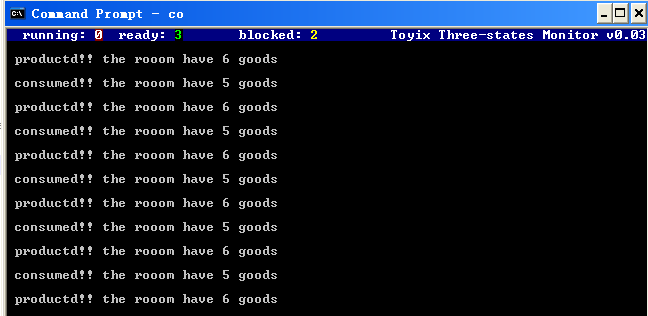
这样生产者和消费者在共享资源的时候就互斥了，不会产生混乱，因为生产者使用的时候消费者不能使用，消费者使用的时候生产者不能使用，所以这样的共享资源就被称为临界资源。

但是我们会发现一个问题，就是生产者，生产一个货物就必须等待消费者把这个货物拿走。但是实际应该是只要缓冲区还能放货物，生产者就放货物。当不能放的时候就必须等待消费者拿走货物。 另一面就是缓冲区的消费者将货物消费完了，就必须等待生产者生产商品，否则就继续拿走货物。我们的程序设计没有体现出来，那怎么设计呢？我们再仔细看了一下程序原来是我们程序中加了delay延时，这样遇到延时就会堵塞生产者进程，执行消费者程序，但是我们将资源上锁了，所以消费者程序不能继续往下执行，但是会登记自己在等待这个资源，所以生产者一释放，消费者程序立马执行。最后就会出现生产者和消费争抢资源，但不会混乱。

下来我们把delay延时函数拿掉来看看效果

结果如下：





会发现最后消费者和生产者速度平衡了，那怎么办呢？解决方法如下：

#include <toyix.h>

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

p(&mutex);

goods++;

printf("\n productd!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

void consumer()

{

while(1)

{

delay(1);

p(&mutex);

goods--;

printf("\n consumed!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

main()

{

set(&mutex,1);

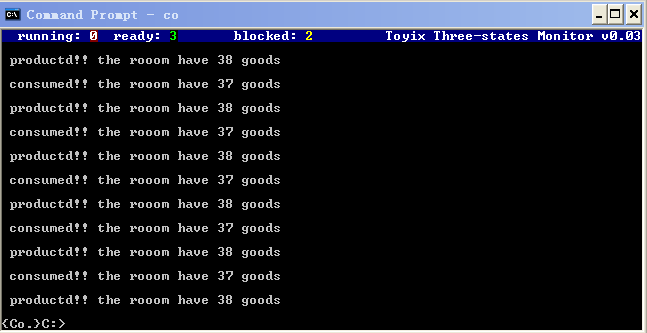
cobegin(producer, consumer,0);

getch();

}

在消费者p前加一个很小的延时，这样消费者消费的速度就没有生产者快啦。

验证结果如下：



可以发现生产者生产速度快了，消费者消费慢了（我给大家截取的是局部，真正效果大家自己去验证更直观），也看到两个进程也没有混乱。

那么让消费者消费快一点，生产者生产慢一点怎么处理呢？

#include <toyix.h>

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

delay(1);

p(&mutex);

goods++;

printf("\n productd!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

void consumer()

{

while(1)

{

p(&mutex);

goods--;

printf("\n consumed!! the rooom have %d goods\n ",goods);

v(&mutex);

}

}

main()

{

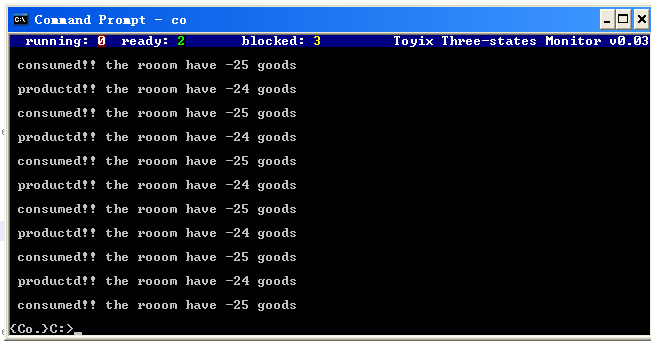
set(&mutex,1);

cobegin(producer, consumer,0);

getch();

}

就是在生产者p前面加个延时，消费者那个去掉。结果如下：



看到变成负的，也就说明消费者消费速度更快喽。

但是现在的问题就是我们不可以让值变成负的（意思就是我们缓冲区为0了就不要再消费了），我们不可以让正值无限的变大（意思就是缓冲区有固定大小，到了最大的时候，就不要继续生产了）。又怎么解决呢？

解决方法如下：

#include <toyix.h>

semaphore empty;

semaphore full;

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

p(&mutex);

p(&empty);

goods++;

printf("\n productd!! the rooom have %d goods\n ",goods);

v(&full);

v(&mutex);

}

}

void consumer()

{

while(1)

{

delay(1);

p(&mutex);

p(&full);

goods--;

printf("\n consumed!! the rooom have %d goods\n ",goods);

v(&empty);

v(&mutex);

}

}

main()

{

set(&empty,10);

set(&full,0);

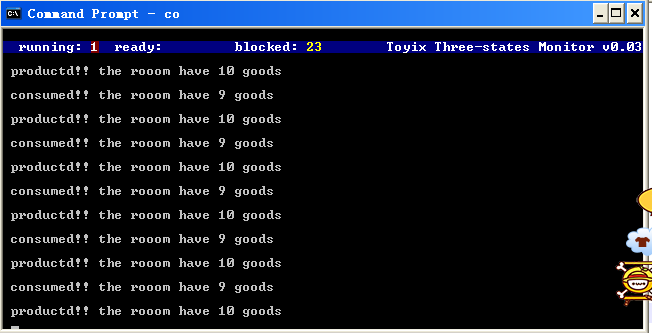
set(&mutex,1);

cobegin(producer, consumer,0);

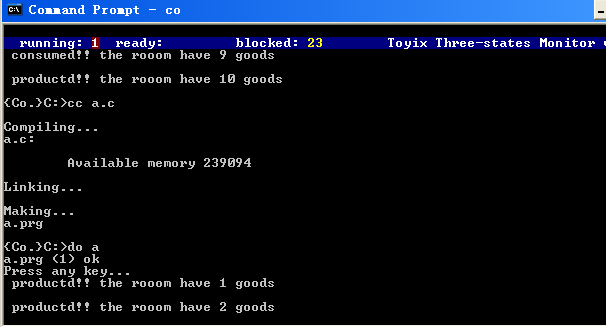
getch();

}

结果让我们很震惊，大家先看结果：



一直卡在10这里卡主了，我试试吧缓冲区大小改成2，结果如下：



一直卡在2这里。在这里就出现了死锁，也就是生产者牢牢的锁住了自己，谁也进不去了。

我们来分析下为什么会产生死锁呢？

开始我们mutex信号量的值为1，empty信号量的值为2，full信号量的值为0；

在生产者程序里先mutex-1，empty-1，然后生产出一个货物,full+1，mutex+1，开始生产者还是比消费者快（为什么这样还不清楚）所以然后又是快速的mutex-1，消费者就又不能执行程序，empty-1,生产出一个货物，full+1,empty+1,这个时候生产者可能还是比较快，就empty-1了，消费者又不能执行程序，但是这个时候empty已经是0了，不再执行下面的程序，标记一下，但是消费者也不能执行，就这样mutex一直就是<1的，empty也是一直小于1，这样两个进程都不能执行，从而产生死锁。

解决方法如下：

#include <toyix.h>

semaphore empty;

semaphore full;

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

p(&empty);

p(&mutex);

goods++;

printf("\n productd!! the rooom have %d goods\n ",goods);

v(&mutex);

v(&full);

}

}

void consumer()

{

while(1)

{

delay(1);

p(&full);

p(&mutex);

goods--;

printf("\n consumed!! the rooom have %d goods\n ",goods);

v(&mutex);

v(&empty);

}

}

main()

{

set(&empty,10);

set(&full,0);

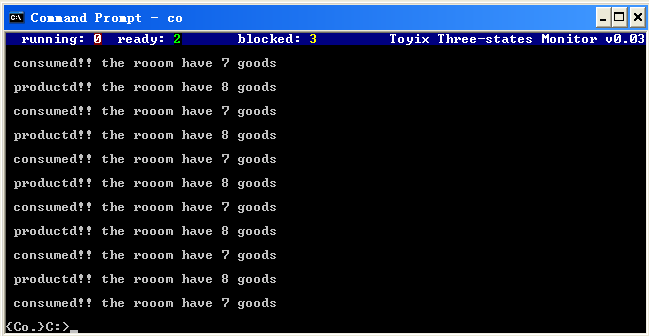
set(&mutex,1);

cobegin(producer, consumer,0);

getch();

}

得到的结果如下：



这样的设计之后就不会发生死锁，而且也有了最大值的限制和最小值得限制。很好的解决了生产者和消费者问题。

最后我们也可以设计多个生产者和多个消费者

设计如下：

#include <toyix.h>

semaphore empty;

semaphore full;

semaphore mutex1;

semaphore mutex2;

semaphore mutex;

int goods=0;

void producer()

{

while(1)

{

p(&empty);

p(&mutex1);

p(&mutex);

goods++;

printf("\n producer%d product one!! the rooom have %d goods\n ",get\_pid(),goods);

v(&mutex);

v(&mutex1);

v(&full);

}

}

void consumer()

{

while(1)

{

delay(1000);

p(&full);

p(&mutex2);

p(&mutex);

goods--;

printf("\n consumer%d take one!! the rooom have %d goods\n ",get\_pid()-4,goods);

v(&mutex);

v(&mutex2);

v(&empty);

}

}

main()

{

set(&empty,40);

set(&full,0);

set(&mutex1,1);

set(&mutex2,1);

set(&mutex,1);

cobegin(producer,0);

cobegin(producer,0);

cobegin(producer,0);

cobegin(producer,0);

cobegin(consumer,0);

cobegin(consumer,0);

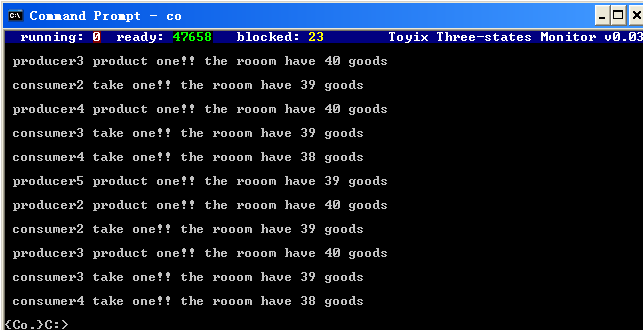
cobegin(consumer,0);

cobegin(consumer,0);

getch();

}

结果如下：



在VC中的设计如下：

#include <windows.h>

#include <stdio.h>

#include <iostream.h>

#include <conio.h>

int goods=0;

HANDLE empty\_Semaphore;//设置信号量用。empty\_Semaphore表示空的缓冲池的数量

HANDLE full\_Semaphore;//用full\_Semaphore表示满的缓冲池的数量

HANDLE mutex1\_Semaphore;//用mutex\_Semaphore表示互斥信号量

HANDLE mutex2\_Semaphore;//用mutex\_Semaphore表示互斥信号量

HANDLE mutex\_Semaphore;

void Producer(void )//创建生产者进程

{

while(1)

{

WaitForSingleObject(empty\_Semaphore,-1);//对empty\_Semaphore进行P操作

WaitForSingleObject(mutex1\_Semaphore,-1);//对mutex\_Semaphore进行P操作

WaitForSingleObject(mutex\_Semaphore,-1);//对mutex\_Semaphore进行P操作

goods++;

printf("\n producer%ld product one!! the rooom have %d goods\n ",GetCurrentThreadId(),goods);

ReleaseSemaphore(mutex\_Semaphore,1,NULL);//对full\_Semaphore进行V操作

ReleaseSemaphore(mutex1\_Semaphore,1,NULL);//对full\_Semaphore进行V操作

ReleaseSemaphore(full\_Semaphore,1,NULL);//对mutex\_Semaphore进行V操作

}

}

void Consumer(void )//创建消费者进程

{

while(1)

{

Sleep(10);

WaitForSingleObject(full\_Semaphore,-1);//对full\_Semaphore进行P操作

WaitForSingleObject(mutex2\_Semaphore,-1);//对mutex2\_Semaphore进行P操作

WaitForSingleObject(mutex\_Semaphore,-1);//对mutex\_Semaphore进行P操作

goods--;

printf("\n consumer%ld take one!! the rooom have %d goods\n ",GetCurrentThreadId(),goods);

ReleaseSemaphore(mutex\_Semaphore,1,NULL);

ReleaseSemaphore(mutex2\_Semaphore,1,NULL);

ReleaseSemaphore(empty\_Semaphore,1,NULL);

}

}

void main ()

{

empty\_Semaphore=CreateSemaphore(NULL,10,10,NULL);//创建信号量empty\_Semaphore

full\_Semaphore=CreateSemaphore(NULL,0,10,NULL);//创建信号量full\_Semaphore

mutex1\_Semaphore=CreateSemaphore(NULL,1,1,NULL); mutex2\_Semaphore=CreateSemaphore(NULL,1,1,NULL);

mutex\_Semaphore=CreateSemaphore(NULL,1,1,NULL);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Producer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Producer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Producer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Producer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Consumer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Consumer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Consumer),0,0,0);

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(Consumer),0,0,0);

getch();

return;

}

运行结果如下：

