问题：以下是生产者与消费者问题的实现，调试代码，发现问题并修改。

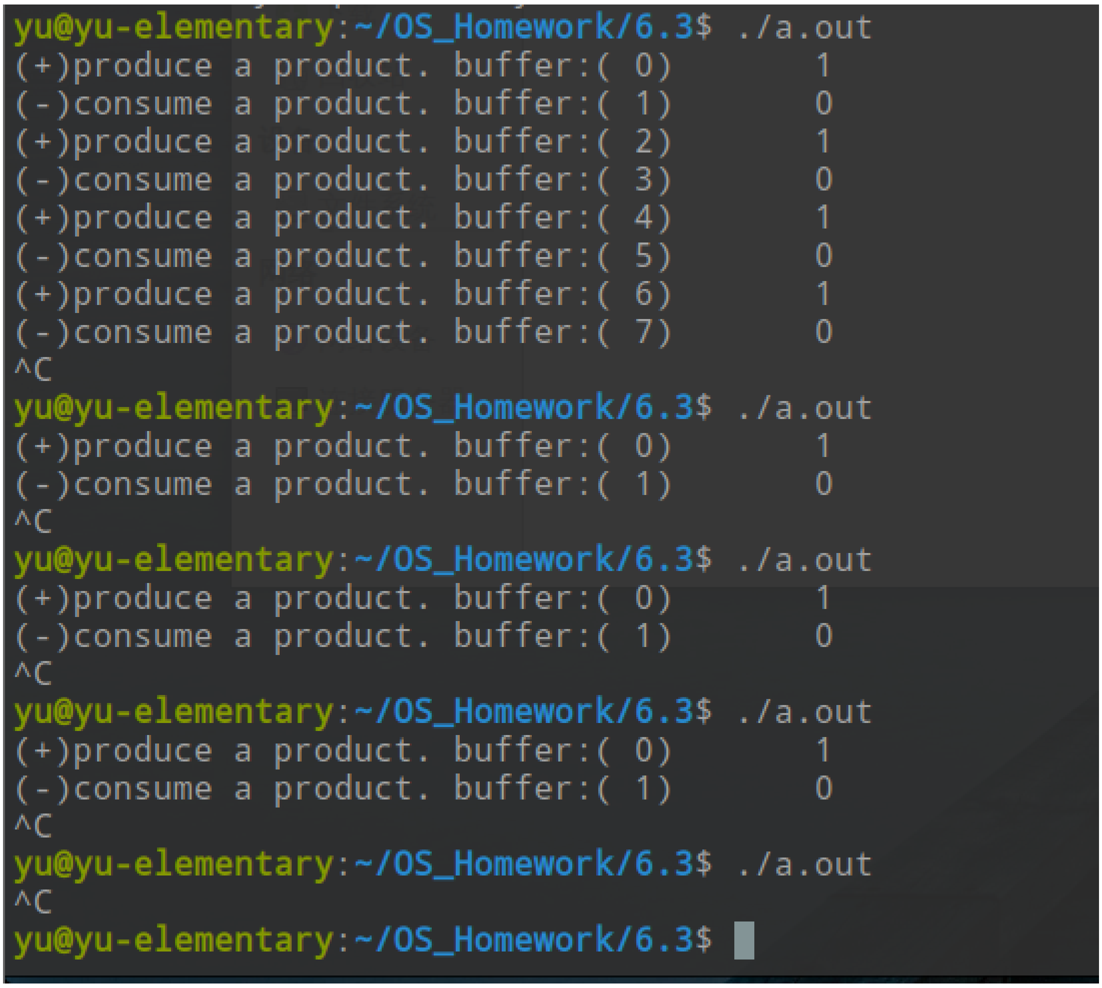
说明：

1. gcc编译时加-lpthread
2. 将调试过程捕捉到错误、修改后正确运行等关建画面截图

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define M 1
#define P(x) sem_wait(&x)
#define V(x) sem_post(&x)
int in = 0;
int out = 0;
int buff[M] = {0};
sem_t sem_dr;
sem_t sem_co;
pthread_mutex_t mutex;

void print() {
    static int number = 0;
    int i;
    printf("(%2d)\t", number);
    for (i = 0; i < M; i++)
        printf("%d ", buff[i]);     //打印buff[0]
    number++;
    printf("\n");
}

void *producer() {
    for (;;) {
        sleep(1);
        P(sem_dr);
        pthread_mutex_lock(&mutex);
        in = in % M;
        printf("(+)produce a product. buffer:");
        buff[in] = 1;
        print();
        ++in;
        pthread_mutex_unlock(&mutex);
        V(sem_co);
    }
}

void *consumer() {
    for (;;) {
        sleep(1);
        pthread_mutex_lock(&mutex);
        P(sem_co);
        out = out % M;
        printf("(-)consume a product. buffer:");
        buff[out] = 0;
        print();
        ++out;
        pthread_mutex_unlock(&mutex);
        V(sem_dr);
    }
}

void sem_mutex_init() {
    int init1 = sem_init(&sem_dr, 0, M);    //初始化信号量sem_dr并设初值为M(1)
    int init2 = sem_init(&sem_co, 0, 0);    //初始化信号量sem_co并设初值为0
    if ((init1 != 0) && (init2 != 0)) {     //判断是否成功（sem_init返回0为成功）
        printf("sem init failed \n");
        exit(1);
    }
    int init3 = pthread_mutex_init(&mutex, NULL);   //初始化mutex
    if (init3 != 0) {                               //判断是否成功
        printf("mutex init failed \n");
        exit(1);
    }
}

int main() {
    pthread_t id1;
    pthread_t id2;
    int i;
    int ret;
    sem_mutex_init();   //初始化信号量与mutex
    /*create the producer thread*/
    ret = pthread_create(&id1, NULL, producer, NULL);
    if (ret != 0) {
        printf("producer creation failed \n");
        exit(1);
    }
    ret = pthread_create(&id2, NULL, consumer, NULL);
    if (ret != 0) {
        printf("consumer creation failed \n");
        exit(1);
    }
    pthread_join(id1, NULL);
    pthread_join(id2, NULL);
    exit(0);
}
```

答：

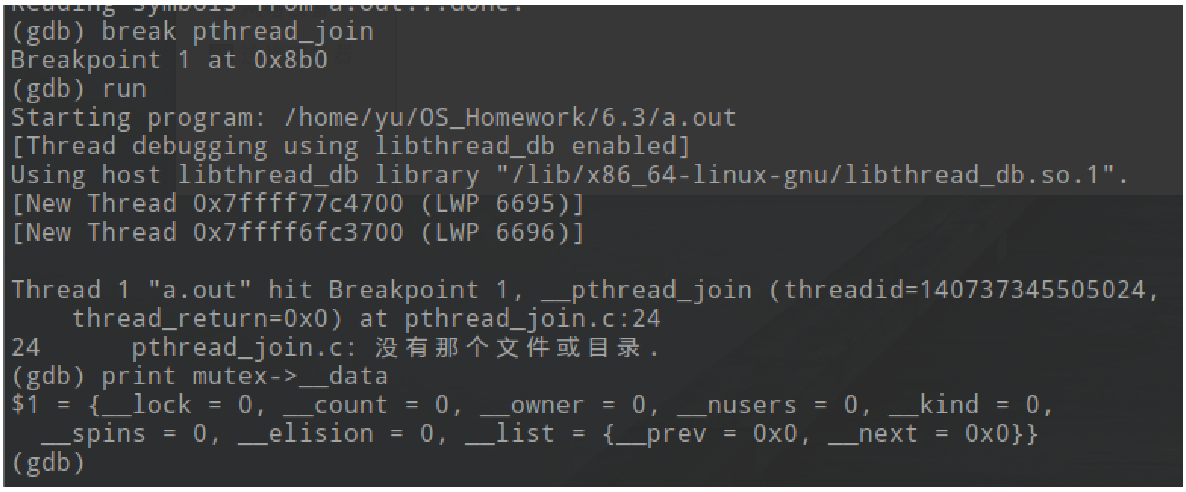
首先运行程序看看效果：



```
yu@yu-elementary:~/OS_Homework/6.3$ ./a.out
(+)produce a product. buffer:( 0)       1
(-)consume a product. buffer:( 1)       0
(+)produce a product. buffer:( 2)       1
(-)consume a product. buffer:( 3)       0
(+)produce a product. buffer:( 4)       1
(-)consume a product. buffer:( 5)       0
(+)produce a product. buffer:( 6)       1
(-)consume a product. buffer:( 7)       0
^C
yu@yu-elementary:~/OS_Homework/6.3$ ./a.out
(+)produce a product. buffer:( 0)       1
(-)consume a product. buffer:( 1)       0
^C
yu@yu-elementary:~/OS_Homework/6.3$ ./a.out
(+)produce a product. buffer:( 0)       1
(-)consume a product. buffer:( 1)       0
^C
yu@yu-elementary:~/OS_Homework/6.3$ ./a.out
(+)produce a product. buffer:( 0)       1
(-)consume a product. buffer:( 1)       0
^C
yu@yu-elementary:~/OS_Homework/6.3$ ./a.out
^C
yu@yu-elementary:~/OS_Homework/6.3$ 
```

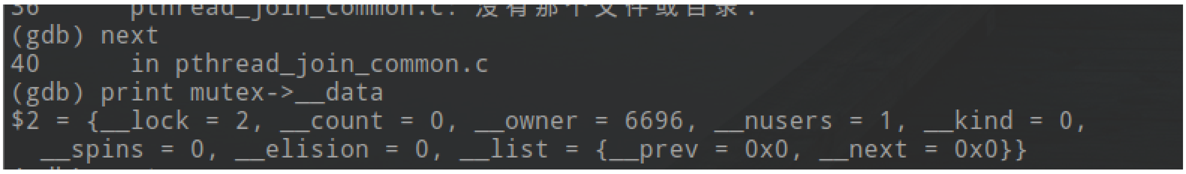
程序会在运行一段时间（有时是刚开始）会停止输出，发生死锁。

使用gdb调试，在线程创建后设置断点，检查互斥锁，未上锁，正常。



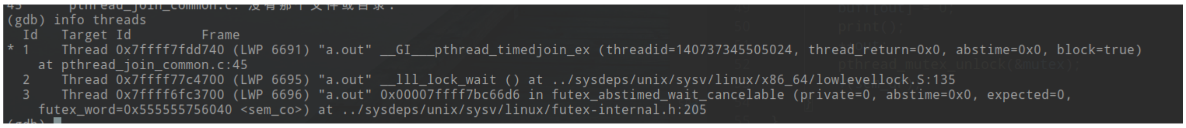
```
Reading symbols from a.out...done.
(gdb) break pthread_join
Breakpoint 1 at 0x8b0
(gdb) run
Starting program: /home/yu/OS_Homework/6.3/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff77c4700 (LWP 6695)]
[New Thread 0x7ffff6fc3700 (LWP 6696)]

Thread 1 "a.out" hit Breakpoint 1, __pthread_join (threadid=140737345505024,
    thread_return=0x0) at pthread_join.c:24
24      pthread_join.c: 没有那个文件或目录.
(gdb) print mutex->__data
$1 = {__lock = 0, __count = 0, __owner = 0, __nusers = 0, __kind = 0,
  __spins = 0, __elision = 0, __list = {__prev = 0x0, __next = 0x0}}
(gdb)
```

继续运行，发现没有输出，检查互斥锁，mutex->__data->__lock已为2，则已经上锁



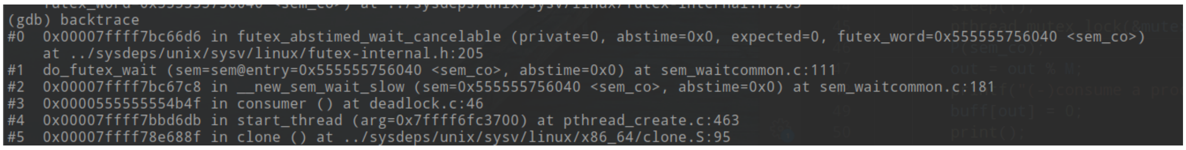
```
36      pthread_join_common.c: 没有那个文件或目录.
(gdb) next
40      in pthread_join_common.c
(gdb) print mutex->__data
$2 = {__lock = 2, __count = 0, __owner = 6696, __nusers = 1, __kind = 0,
  __spins = 0, __elision = 0, __list = {__prev = 0x0, __next = 0x0}}
```

可以看到互斥锁被6696号进程（线程）所占有



```
45      pthread_join_common.c: 没有那个文件或目录.
(gdb) info threads
  Id   Target Id         Frame
* 1    Thread 0x7ffff7fdd740 (LWP 6691) "a.out" __GI___pthread_timedjoin_ex (threadid=140737345505024, thread_return=0x0, abstime=0x0, block=true)
    at pthread_join_common.c:45
  2    Thread 0x7ffff77c4700 (LWP 6695) "a.out" __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
  3    Thread 0x7ffff6fc3700 (LWP 6696) "a.out" 0x00007ffff7bc66d6 in futex_abstimed_wait_cancelable (private=0, abstime=0x0, expected=0,
    futex_word=0x555555756040 <sem_co>) at ../sysdeps/unix/sysv/linux/futex-internal.h:205
(gdb) 
```

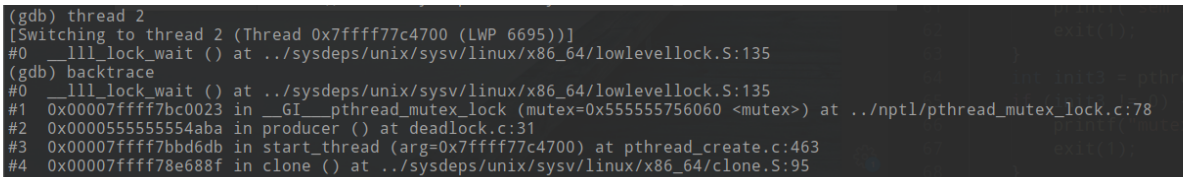
可以看到6696的线程id为3，使用thread 3切换至该线程，再用backtrace



```
    futex_word=0x555555756040 <sem_co>) at ../sysdeps/unix/sysv/linux/futex-internal.h:205
(gdb) backtrace
#0  0x00007ffff7bc66d6 in futex_abstimed_wait_cancelable (private=0, abstime=0x0, expected=0, futex_word=0x555555756040 <sem_co>)
    at ../sysdeps/unix/sysv/linux/futex-internal.h:205
#1  do_futex_wait (sem=sem@entry=0x555555756040 <sem_co>, abstime=0x0) at sem_waitcommon.c:111
#2  0x00007ffff7bc67c8 in __new_sem_wait_slow (sem=0x555555756040 <sem_co>, abstime=0x0) at sem_waitcommon.c:181
#3  0x0000555555554b4f in consumer () at deadlock.c:46
#4  0x00007ffff7bbd6db in start_thread (arg=0x7ffff6fc3700) at pthread_create.c:463
#5  0x00007ffff78e688f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

发现线程卡在了第46行，即消费者线程P(sem_co);处，可见消费者进程已取得了互斥锁，正等待生产者生产完后唤醒它。

此时切回线程2，使用backtrace追踪



```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7ffff77c4700 (LWP 6695))]
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
(gdb) backtrace
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
#1  0x00007ffff7bc0023 in __GI___pthread_mutex_lock (mutex=0x555555756060 <mutex>) at ../nptl/pthread_mutex_lock.c:78
#2  0x0000555555554aba in producer () at deadlock.c:31
#3  0x00007ffff7bbd6db in start_thread (arg=0x7ffff77c4700) at pthread_create.c:463
#4  0x00007ffff78e688f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

发现卡在了第31行，即生产者的pthread_mutex_lock(&mutex);中，可见生产者并没有能力生产消费者所需的产品，因为它需要等待消费者为其解互斥锁才能继续进行。此时死锁就发生了。

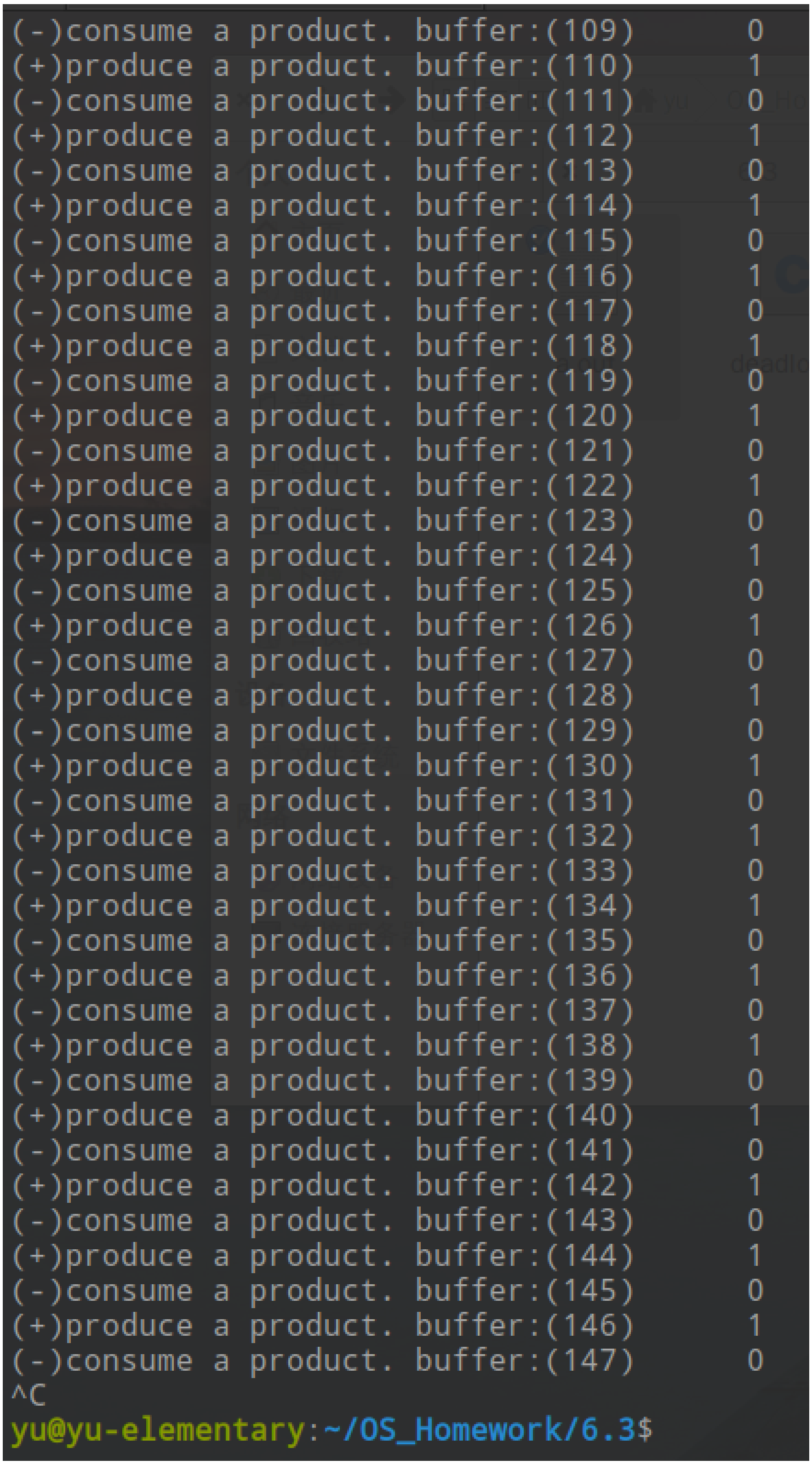
死锁是互斥锁与信号量操作的先后顺序问题导致的，检查源代码，发现消费者的互斥锁上锁过程在P原语之前，将它移至P原语之后（即上述代码中45、46行调换位置），程序恢复正常。

最终程序：

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <unistd.h>
 4  #include <pthread.h>
 5  #include <semaphore.h>
 6
 7  #define M 1
 8  #define P(x) sem_wait(&x)
 9  #define V(x) sem_post(&x)
10  int in = 0;
11  int out = 0;
12  int buff[M] = {0};
13  sem_t sem_dr;
14  sem_t sem_co;
15  pthread_mutex_t mutex;
16
17  void print() {
18      static int number = 0;
19      int i;
20      printf("(%2d)\t", number);
21      for (i = 0; i < M; i++)
22          printf("%d ", buff[i]);     //打印buff[0]
23      number++;
24      printf("\n");
25  }
26
27  void *producer() {
28      for (;;) {
29          sleep(1);
30          P(sem_dr);
31          pthread_mutex_lock(&mutex);
32          in = in % M;
33          printf("(+)produce a product. buffer:");
34          buff[in] = 1;
35          print();
36          ++in;
37          pthread_mutex_unlock(&mutex);
38          V(sem_co);
```

```
    }
}

void *consumer() {
    for (;;) {
        sleep(1);
        P(sem_co);
        pthread_mutex_lock(&mutex);
        out = out % M;
        printf("(-)consume a product. buffer:");
        buff[out] = 0;
        print();
        ++out;
        pthread_mutex_unlock(&mutex);
        V(sem_dr);
    }
}

void sem_mutex_init() {
    int init1 = sem_init(&sem_dr, 0, M);    //初始化信号量sem_dr并设初值为M(1)
    int init2 = sem_init(&sem_co, 0, 0);    //初始化信号量sem_co并设初值为0
    if ((init1 != 0) && (init2 != 0)) {     //判断是否成功（sem_init返回0为成功）
        printf("sem init failed \n");
        exit(1);
    }
    int init3 = pthread_mutex_init(&mutex, NULL);   //初始化mutex
    if (init3 != 0) {                               //判断是否成功
        printf("mutex init failed \n");
        exit(1);
    }
}

int main() {
    pthread_t id1;
    pthread_t id2;
    int i;
    int ret;
    sem_mutex_init();   //初始化信号量与mutex
    /*create the producer thread*/
    ret = pthread_create(&id1, NULL, producer, NULL);
    if (ret != 0) {
        printf("producer creation failed \n");
        exit(1);
    }
    ret = pthread_create(&id2, NULL, consumer, NULL);
    if (ret != 0) {
        printf("consumer creation failed \n");
        exit(1);
    }
    pthread_join(id1, NULL);
    pthread_join(id2, NULL);
    exit(0);
}
```

执行结果:



```
(-)consume a product. buffer:(109)    0
(+)produce a product. buffer:(110)    1
(-)consume a product. buffer:(111)    0
(+)produce a product. buffer:(112)    1
(-)consume a product. buffer:(113)    0
(+)produce a product. buffer:(114)    1
(-)consume a product. buffer:(115)    0
(+)produce a product. buffer:(116)    1
(-)consume a product. buffer:(117)    0
(+)produce a product. buffer:(118)    1
(-)consume a product. buffer:(119)    0
(+)produce a product. buffer:(120)    1
(-)consume a product. buffer:(121)    0
(+)produce a product. buffer:(122)    1
(-)consume a product. buffer:(123)    0
(+)produce a product. buffer:(124)    1
(-)consume a product. buffer:(125)    0
(+)produce a product. buffer:(126)    1
(-)consume a product. buffer:(127)    0
(+)produce a product. buffer:(128)    1
(-)consume a product. buffer:(129)    0
(+)produce a product. buffer:(130)    1
(-)consume a product. buffer:(131)    0
(+)produce a product. buffer:(132)    1
(-)consume a product. buffer:(133)    0
(+)produce a product. buffer:(134)    1
(-)consume a product. buffer:(135)    0
(+)produce a product. buffer:(136)    1
(-)consume a product. buffer:(137)    0
(+)produce a product. buffer:(138)    1
(-)consume a product. buffer:(139)    0
(+)produce a product. buffer:(140)    1
(-)consume a product. buffer:(141)    0
(+)produce a product. buffer:(142)    1
(-)consume a product. buffer:(143)    0
(+)produce a product. buffer:(144)    1
(-)consume a product. buffer:(145)    0
(+)produce a product. buffer:(146)    1
(-)consume a product. buffer:(147)    0
^C
yu@yu-elementary:~/OS_Homework/6.3$
```

程序可长期稳定执行。