1 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）

deadlock.c

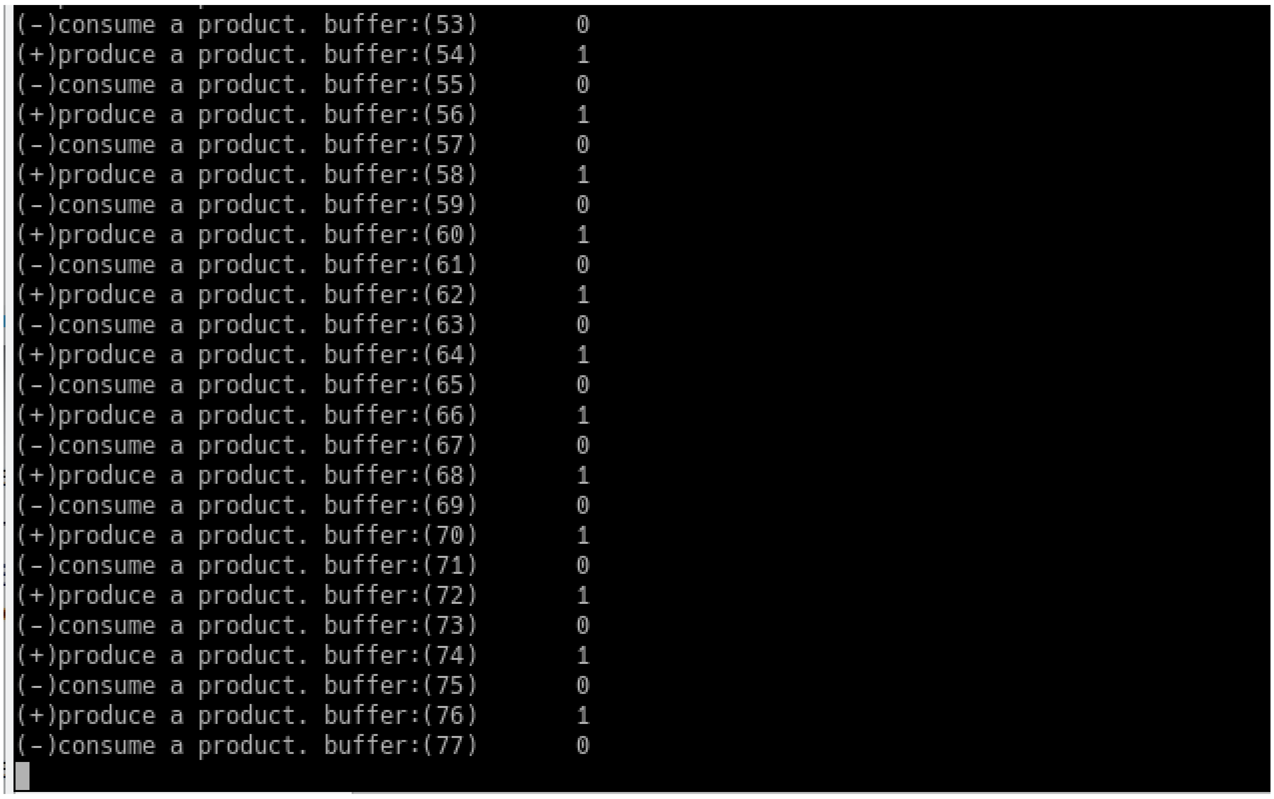
正确答案：

学生答案：

编译：

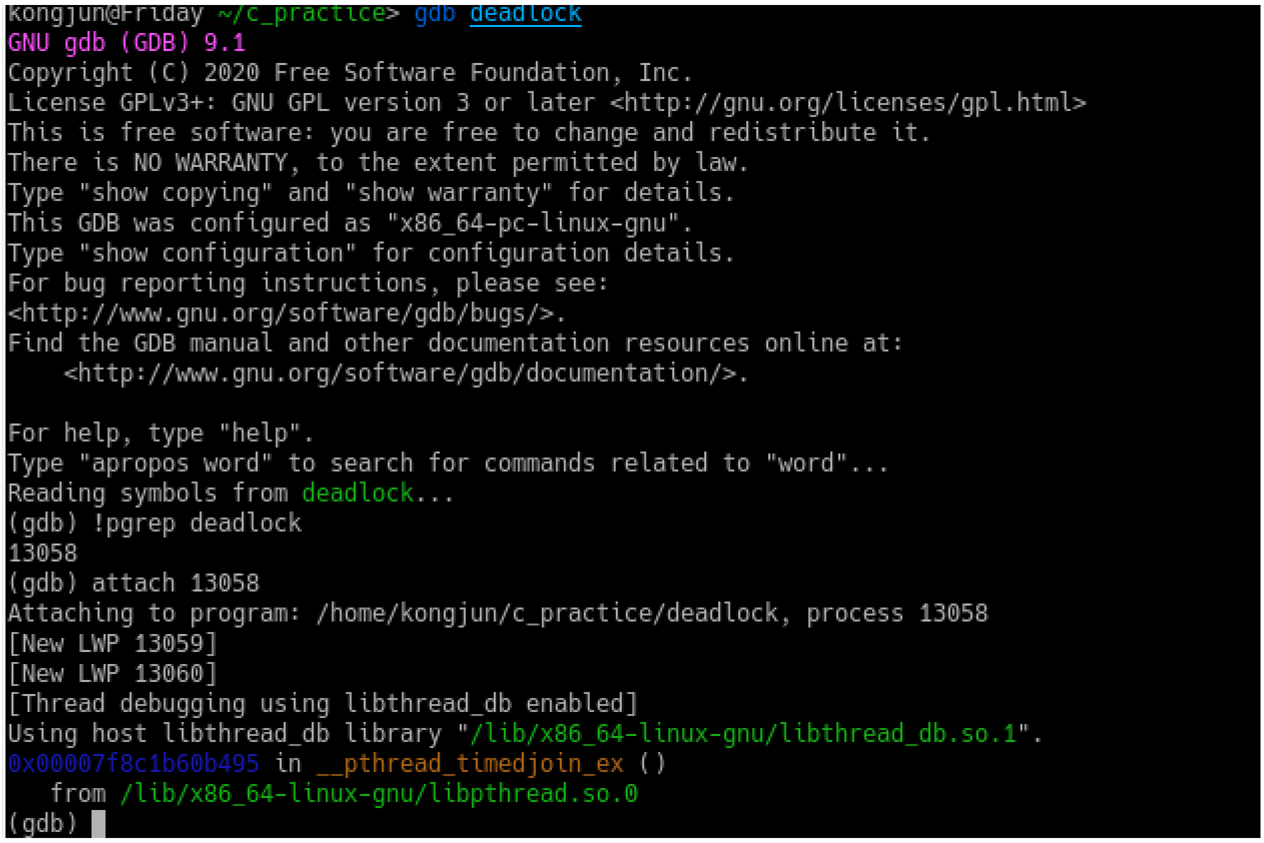
gcc -Wall -gdwarf-4 -pthread -o deadlock deadlock.c

执行一段时间后程序卡住。如图：



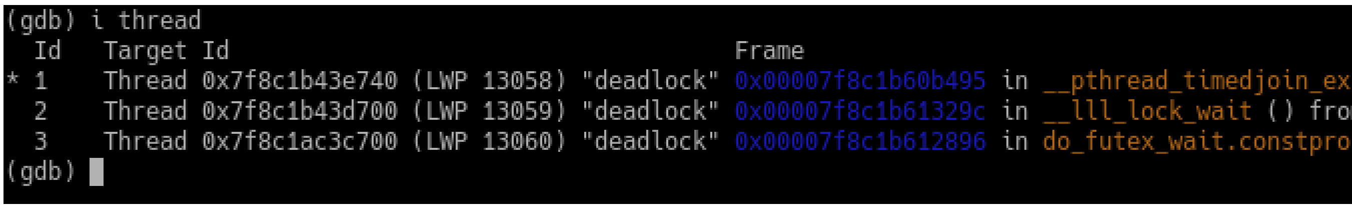
```
(-)consume a product. buffer:(53)      0
(+)produce a product. buffer:(54)      1
(-)consume a product. buffer:(55)      0
(+)produce a product. buffer:(56)      1
(-)consume a product. buffer:(57)      0
(+)produce a product. buffer:(58)      1
(-)consume a product. buffer:(59)      0
(+)produce a product. buffer:(60)      1
(-)consume a product. buffer:(61)      0
(+)produce a product. buffer:(62)      1
(-)consume a product. buffer:(63)      0
(+)produce a product. buffer:(64)      1
(-)consume a product. buffer:(65)      0
(+)produce a product. buffer:(66)      1
(-)consume a product. buffer:(67)      0
(+)produce a product. buffer:(68)      1
(-)consume a product. buffer:(69)      0
(+)produce a product. buffer:(70)      1
(-)consume a product. buffer:(71)      0
(+)produce a product. buffer:(72)      1
(-)consume a product. buffer:(73)      0
(+)produce a product. buffer:(74)      1
(-)consume a product. buffer:(75)      0
(+)produce a product. buffer:(76)      1
(-)consume a product. buffer:(77)      0
```

用 GDB attach 到该进程，查看运行情况。



```
kongjun@Friday ~/c_practice> gdb deadlock
GNU gdb (GDB) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from deadlock...
(gdb) !pgrep deadlock
13058
(gdb) attach 13058
Attaching to program: /home/kongjun/c_practice/deadlock, process 13058
[New LWP 13059]
[New LWP 13060]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0x00007f8c1b60b495 in __pthread_timedjoin_ex ()
   from /lib/x86_64-linux-gnu/libpthread.so.0
(gdb) 
```

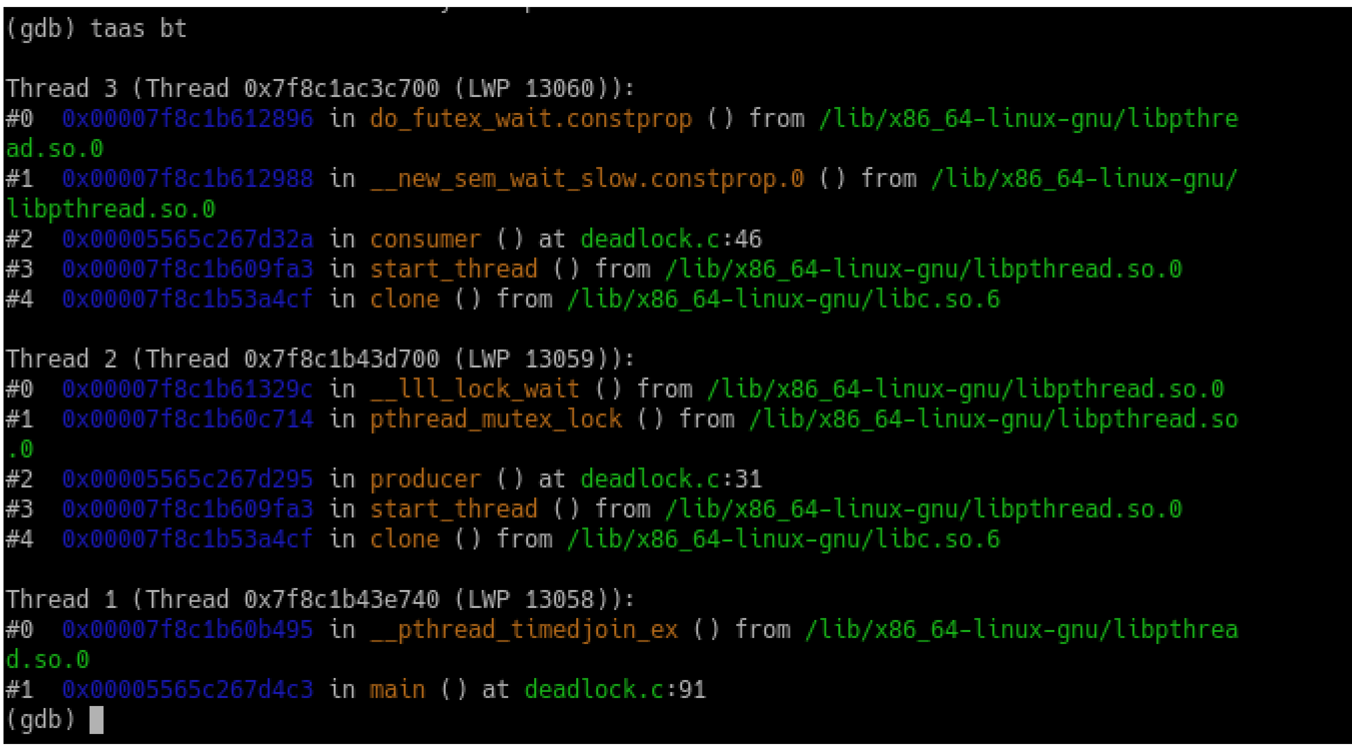
查看进程中的线程信息：



```
(gdb) i thread
  Id   Target Id                                   Frame
* 1    Thread 0x7f8c1b43e740 (LWP 13058) "deadlock" 0x00007f8c1b60b495 in __pthread_timedjoin_ex
  2    Thread 0x7f8c1b43d700 (LWP 13059) "deadlock" 0x00007f8c1b61329c in __lll_lock_wait () fro
  3    Thread 0x7f8c1ac3c700 (LWP 13060) "deadlock" 0x00007f8c1b612896 in do_futex_wait.constpro
(gdb) 
```

发现有三个线程在运行。通过查看堆栈信息，判断这三个线程对应程序中的哪个

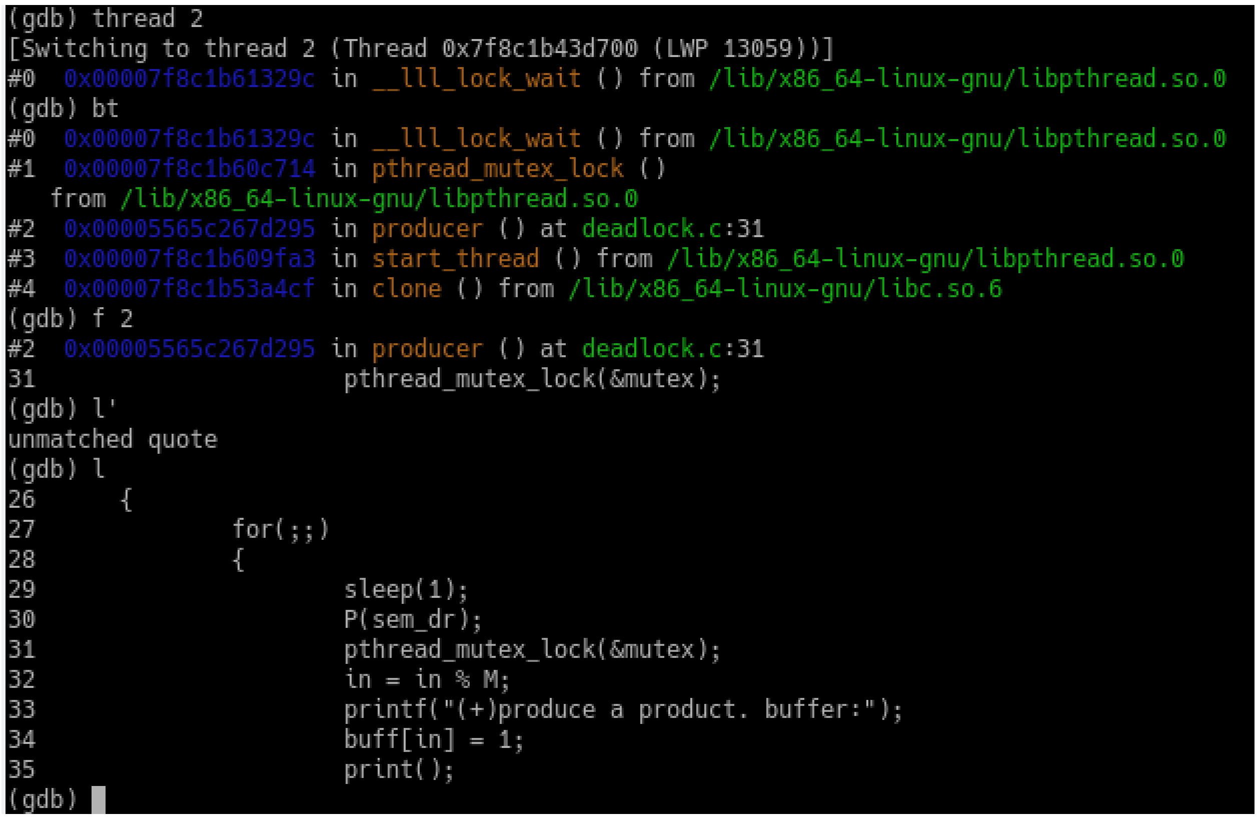
线程。



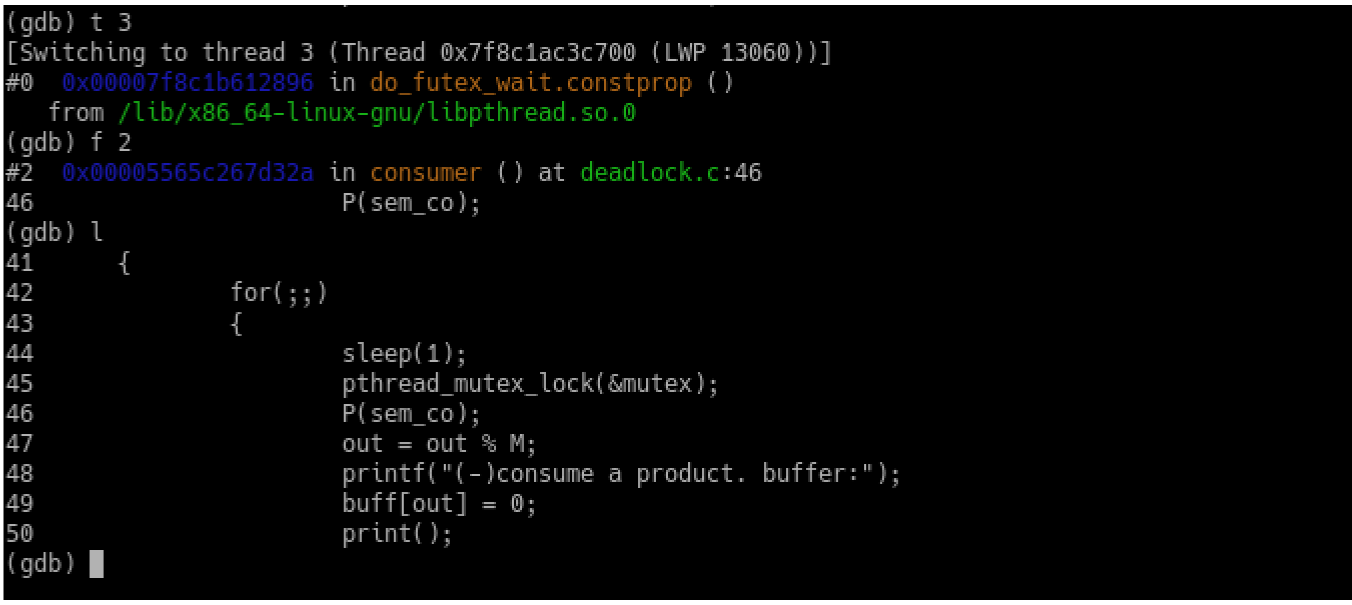
```
(gdb) taas bt

Thread 3 (Thread 0x7f8c1ac3c700 (LWP 13060)):
#0  0x00007f8c1b612896 in do_futex_wait.constprop () from /lib/x86_64-linux-gnu/libpthre
ad.so.0
#1  0x00007f8c1b612988 in __new_sem_wait_slow.constprop.0 () from /lib/x86_64-linux-gnu/
libpthread.so.0
#2  0x00005565c267d32a in consumer () at deadlock.c:46
#3  0x00007f8c1b609fa3 in start_thread () from /lib/x86_64-linux-gnu/libpthread.so.0
#4  0x00007f8c1b53a4cf in clone () from /lib/x86_64-linux-gnu/libc.so.6

Thread 2 (Thread 0x7f8c1b43d700 (LWP 13059)):
#0  0x00007f8c1b61329c in __lll_lock_wait () from /lib/x86_64-linux-gnu/libpthread.so.0
#1  0x00007f8c1b60c714 in pthread_mutex_lock () from /lib/x86_64-linux-gnu/libpthread.so
.0
#2  0x00005565c267d295 in producer () at deadlock.c:31
#3  0x00007f8c1b609fa3 in start_thread () from /lib/x86_64-linux-gnu/libpthread.so.0
#4  0x00007f8c1b53a4cf in clone () from /lib/x86_64-linux-gnu/libc.so.6

Thread 1 (Thread 0x7f8c1b43e740 (LWP 13058)):
#0  0x00007f8c1b60b495 in __pthread_timedjoin_ex () from /lib/x86_64-linux-gnu/libpthrea
d.so.0
#1  0x00005565c267d4c3 in main () at deadlock.c:91
(gdb) 
```

通过堆栈上的函数信息，判断出线程 1 是主线程，线程 2 是生产者线程，线程 3 是消费者进程。

再将栈帧切换到 producer()和 consumer()，查看两个线程阻塞在哪个位置。



```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7f8c1b43d700 (LWP 13059))]
#0  0x00007f8c1b61329c in __lll_lock_wait () from /lib/x86_64-linux-gnu/libpthread.so.0
(gdb) bt
#0  0x00007f8c1b61329c in __lll_lock_wait () from /lib/x86_64-linux-gnu/libpthread.so.0
#1  0x00007f8c1b60c714 in pthread_mutex_lock ()
   from /lib/x86_64-linux-gnu/libpthread.so.0
#2  0x00005565c267d295 in producer () at deadlock.c:31
#3  0x00007f8c1b609fa3 in start_thread () from /lib/x86_64-linux-gnu/libpthread.so.0
#4  0x00007f8c1b53a4cf in clone () from /lib/x86_64-linux-gnu/libc.so.6
(gdb) f 2
#2  0x00005565c267d295 in producer () at deadlock.c:31
31                      pthread_mutex_lock(&mutex);
(gdb) l'
unmatched quote
(gdb) l
26      {
27              for(;;)
28              {
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();
(gdb) 
```



```
(gdb) t 3
[Switching to thread 3 (Thread 0x7f8c1ac3c700 (LWP 13060))]
#0  0x00007f8c1b612896 in do_futex_wait.constprop ()
   from /lib/x86_64-linux-gnu/libpthread.so.0
(gdb) f 2
#2  0x00005565c267d32a in consumer () at deadlock.c:46
46                      P(sem_co);
(gdb) l
41      {
42              for(;;)
43              {
44                      sleep(1);
45                      pthread_mutex_lock(&mutex);
46                      P(sem_co);
47                      out = out % M;
48                      printf("(-)consume a product. buffer:");
49                      buff[out] = 0;
50                      print();
(gdb) 
```

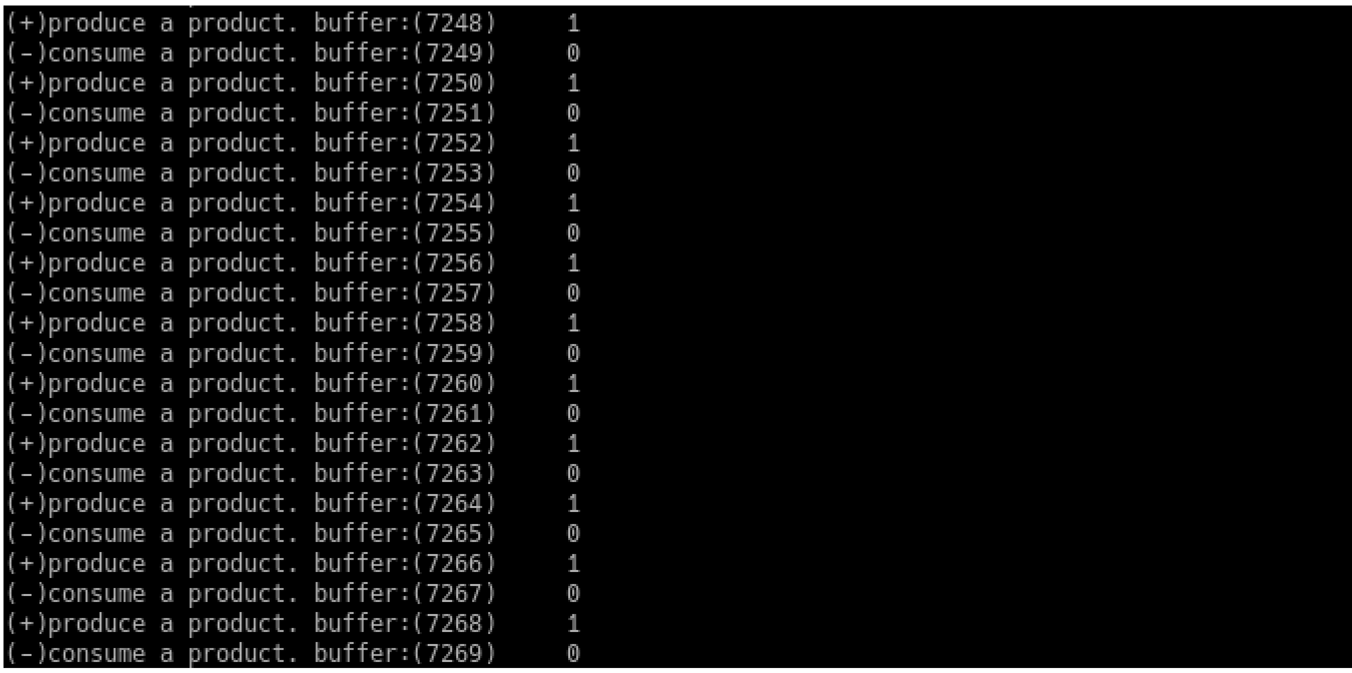
生产者线程阻塞在 pthread_mutex_lock(&mutex)，消费者线程阻塞在 P(sem_co)。

这就发现了程序卡住的原因。生产者线程取得了互斥锁，阻塞在 P(sem_co)，消费者线程阻塞在互斥锁上，无法修改信号量，导致死锁。

为了解决死锁问题，只需要生产者线程先 P(sem_co），再获取互斥锁。具体操作为：将 consumer 函数中 pthread_mutex_lock(&mutex)语句和 P(sem_co)交换次序。

虽然修改后的程序是正确的，但是线程执行次序不确定，死锁可能在任何时候发生，修改后的程序暂时没发生死锁不能说明程序没有死锁的风险，无法用运行结果说明程序正确。

修改后运行结果：



```
(+)produce a product. buffer:(7248)     1
(-)consume a product. buffer:(7249)     0
(+)produce a product. buffer:(7250)     1
(-)consume a product. buffer:(7251)     0
(+)produce a product. buffer:(7252)     1
(-)consume a product. buffer:(7253)     0
(+)produce a product. buffer:(7254)     1
(-)consume a product. buffer:(7255)     0
(+)produce a product. buffer:(7256)     1
(-)consume a product. buffer:(7257)     0
(+)produce a product. buffer:(7258)     1
(-)consume a product. buffer:(7259)     0
(+)produce a product. buffer:(7260)     1
(-)consume a product. buffer:(7261)     0
(+)produce a product. buffer:(7262)     1
(-)consume a product. buffer:(7263)     0
(+)produce a product. buffer:(7264)     1
(-)consume a product. buffer:(7265)     0
(+)produce a product. buffer:(7266)     1
(-)consume a product. buffer:(7267)     0
(+)produce a product. buffer:(7268)     1
(-)consume a product. buffer:(7269)     0
```