警示：本报告两人一组，共同完成。报告以A4规格的PDF格式文档按规定命名并按时提交，严禁抄袭。

**邮件主题**命名：《软件测试》期末大作业\_学号1\_学号2

**邮件附件**命名：《软件测试》期末大作业\_学号1\_学号2.pdf

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分组编号 | **9** | 学号1 | **17343104** | 姓名1 | **唐婵** |
| 学号2 | **17343105** | 姓名2 | **田皓** |
| 提交邮箱 | isscgy@163.com | | 截止提交时间 | 2020年07月31日23时 | |

**【测试材料】**

* 一个用于演示操作系统进程互斥与同步的生产者-消费者模型的示范系统。
* 功能说明：
  1. 启动主控程序 syn-pc-con-6，运行参数是一个可以共享的文件名例如 /home/myshm。该文件必须事先建立。
  2. 主控程序输入临时存储产品的缓冲区大小、计划生产的产品总数量、生产者线程数量和消费者线程数量，并激活生产者和消费者进程。
  3. 生产者进程：若干生产线程向上述缓冲区放入产品（忽略具体的生产过程），缓冲区满时需要等待。生产者进程在完成计划产品总数量时结束。
  4. 消费者进程：若干消费线程从上述缓冲区取出产品，缓冲区空时需要等待。消费者进程在发现生产者进程结束时，取空缓冲区的产品后结束。
* C 源代码清单：

syn-pc-con-6.h

syn-pc-con-6.c

syn-pc-producer-6.c

syn-pc-consumer-6.c

* 某次运行的屏幕快照：

screensnap.png

* 运行平台：x86-PC / Ubuntu 18.04
* 编译器：gcc version 7.5.0
* 运行库：POSIX pthread （编译选项 -lpthread）

**【测试内容】**

1. 静态测试：
   1. 对源代码以你的观点进行静态代码检查，给出检查报告。
2. 复杂性分析：
   1. 计算 syn-pc-con-6.c 的 *Hastead* 复杂度；
   2. 计算 syn-pc-consumer-6.c 的 *McCabe* 复杂度。
3. 白盒测试：
   1. 对 syn-pc-consumer-6.c 实现条件覆盖测试。
4. 黑盒测试：
   1. 对主控模块 syn-pc-con-6.c 的输入实现等价类划分测试。
5. 系统测试：
   1. 自行选择两种故障模型进行软件故障静态注入测试。

**【测试环境】**

操作系统：CentOS 7

编译器：gcc version 7.5.0

运行库：POSIX pthread

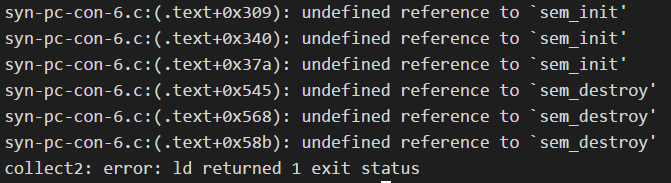
**【测试用例】**

**见测试过程的黑盒测试和白盒测试部分**

**【测试过程】**

**静态测试**

直接尝试编译：gcc syn-pc-con-6.c -o test

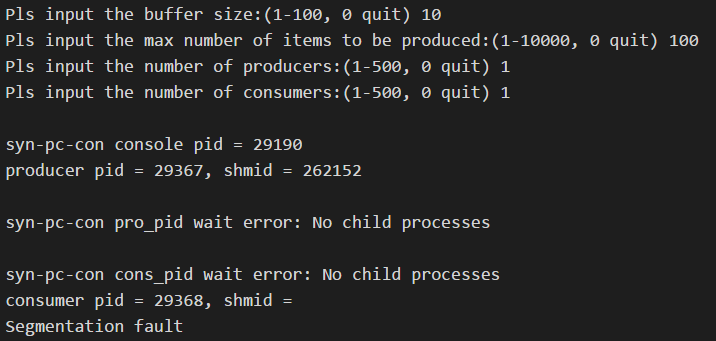
报错：找不到 syn-pc-con-5.h ，查看代码发现#include的文件名错误，改为 syn-pc-con-6.h，再次执行，报错：

因为 pthread 并非默认Linux默认链接库，需要显示链接，即编译命令加上参数 -lpthread ，发现作业要求中【测试材料】部分有提及。

gcc -lpthread syn-pc-con-6.c -o test

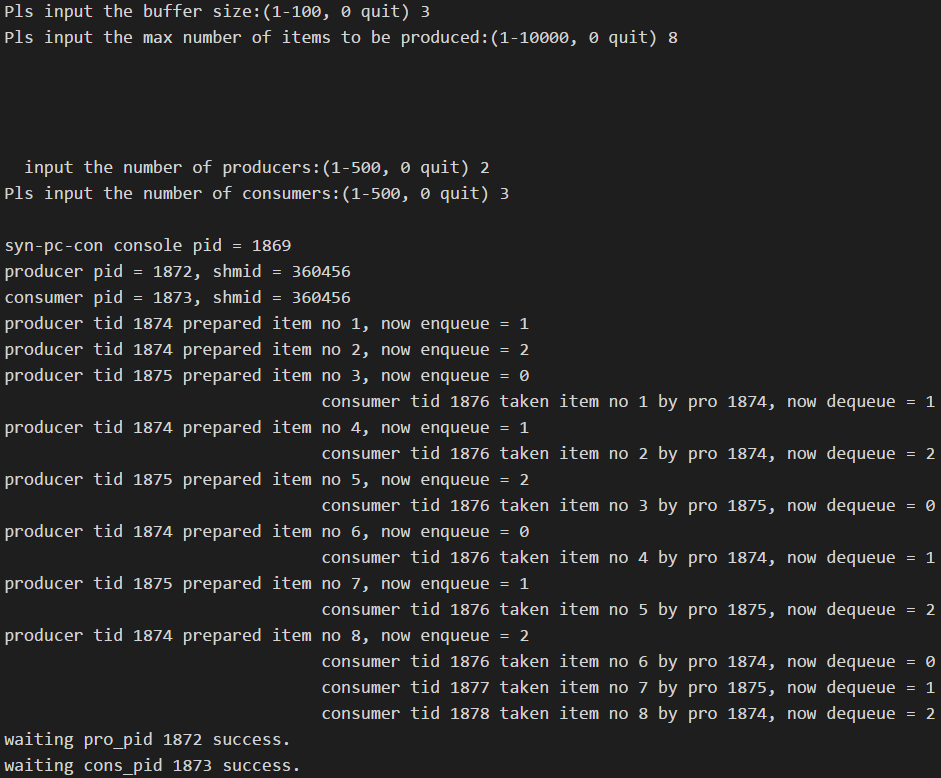
编译成功，创建空目录mytest，运行 ./test mytest 没有反应，查看代码发现文件名写死，再次修改编译命令

gcc -lpthread syn-pc-con-6.c -o syn-pc-con-6.o

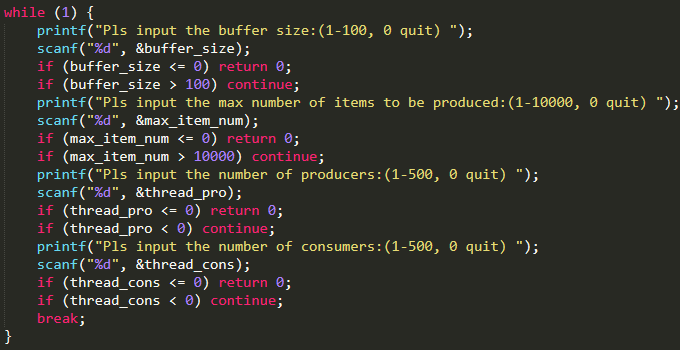
编译成功，运行 ./syn-pc-con-6.o mytest：

段错误，查看源代码，发现需要调用另外两个文件编译生成的程序，于是执行编译

gcc -lpthread syn-pc-consumer-6.c -o syn-pc-consumer-6.o  
gcc -lpthread syn-pc-producer-6.c -o syn-pc-producer-6.o

再次运行 ./syn-pc-con-6.o mytest，得到截图中结果：

随后，观察源代码的输入参数处理部分，发现thread\_pro<0和thread\_cons<0 的判断是冗余的，在continue之前就会return，同时此处的错误会对之后的测试有影响，因为没有判断这两个参数的右边界。



**复杂性分析**

a) 计算 syn-pc-con-6.c 的 *Hastead* 复杂度；

|  |  |  |  |
| --- | --- | --- | --- |
| 操作符 | 数量 | 操作数 | 数量 |
| if | 30 | childpid | 7 |
| <= | 4 | pro\_pid | 8 |
| < | 6 | cons\_pid | 8 |
| == | 13 | statbuf | 2 |
| & | 11 | buffer\_size | 10 |
| while | 1 | max*item*num | 7 |
| return | 14 | thread\_pro | 7 |
| continue | 4 | thread\_cons | 7 |
| break | 1 | argc | 2 |
| = | 36 | argv | 11 |
| -> | 16 | EXIT\_FAILURE | 7 |
| > | 2 | ctln | 20 |
| else | 5 | data | 8 |
| != | 2 | key | 7 |
| - | 13 | ret | 13 |
|  |  | shmid | 13 |
|  |  | 0 | 32 |
|  |  | 1 | 14 |
|  |  | 2 | 3 |
|  |  | 10 | 7 |
|  |  | 3 | 1 |
|  |  | 0x28 | 1 |
| n1 = 15 | N1 = 158 | n2 = 21 | N2 = 185 |

Program vocabulary：n = + = 36

Program length: N = + = 343

Calculated program length: N^ = + = 150.84

Program volume: V = N = 1773.28

Program level: L^ = (2/) \* () = 0.015

Program difficulty：D = 1 / L^ = 66.67

Program Effort：E = V \* D = 118218.67

Language level: L’ = \* V = 0.399

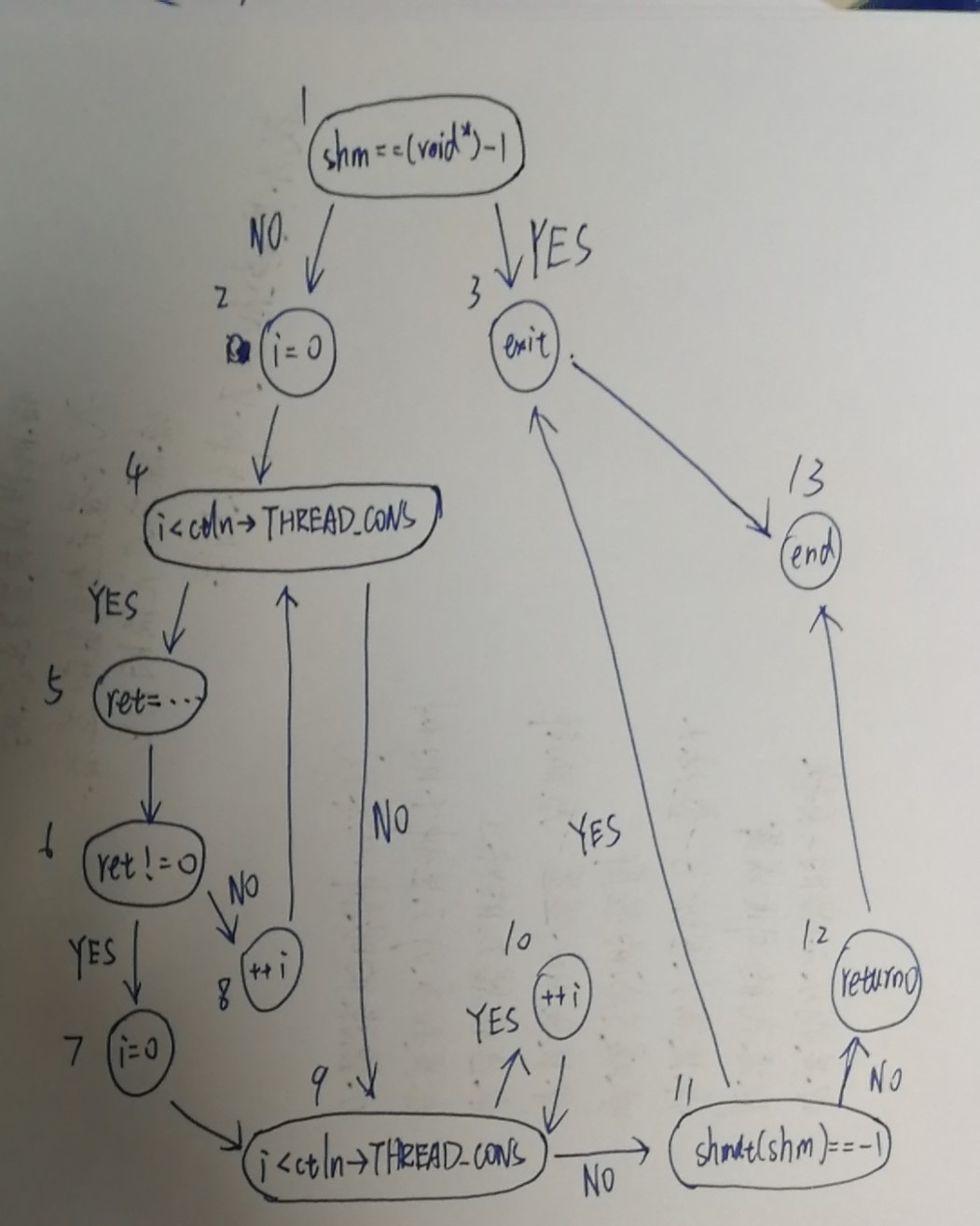
Program Time (hours): T^ = E / (S \* f) = 1.82

平均语句大小：N / 语句数 = 2.98

程序中的错误数预测值：B = V / 3000 = 39.4

b) 计算 syn-pc-consumer-6.c 的 *McCabe* 复杂度。

程序控制流图如下：



McCabe复杂度：V(G) = d +1 = 5 + 1 = 6， d为单条件判断节点个数

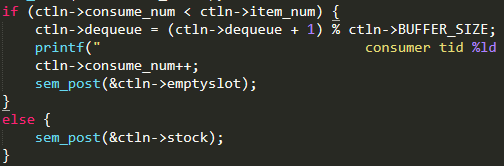
**白盒测试**

a) 对 syn-pc-consumer-6.c 实现条件覆盖测试。

条件覆盖测试即对源代码中每个条件表达式的True和False各取值一次。

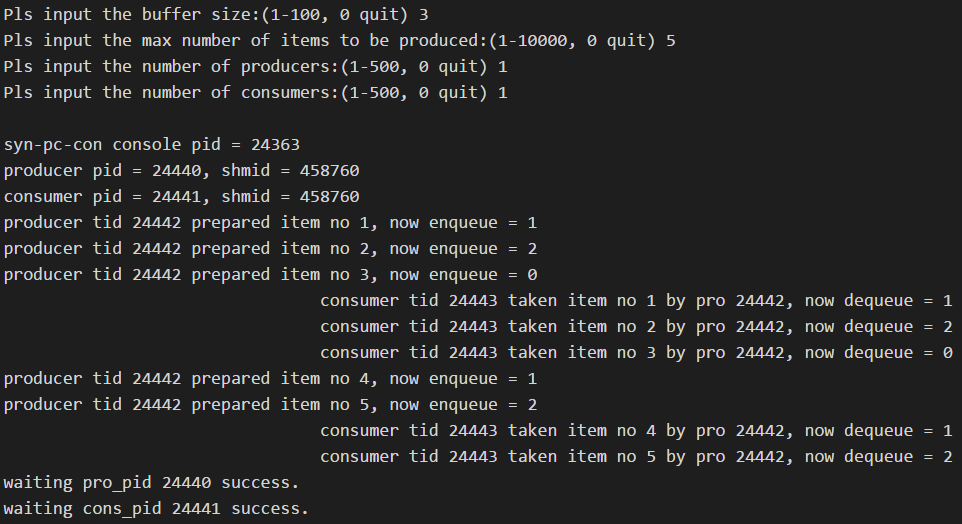
阅读syn-pc-consumer-6.c源代码，判断条件如下：

1. if (ctln->consume\_num < ctln->item\_num)
2. if (shm == (void \*)-1)
3. if (ret != 0)
4. if (shmdt(shm) == -1)

首先考虑覆盖条件1，相关参数有两个，已消费产品总数量和已生产产品总数量

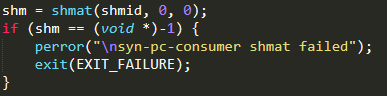
若判断为True则deuqeue移动且消费数加1，False则什么都不做，设计测试用例如下：

|  |  |  |  |
| --- | --- | --- | --- |
| BUFFER\_SIZE | MAX\_ITEM\_NUM | THREAD\_PRO | THREAD\_CONS |
| 3 | 5 | 1 | 1 |

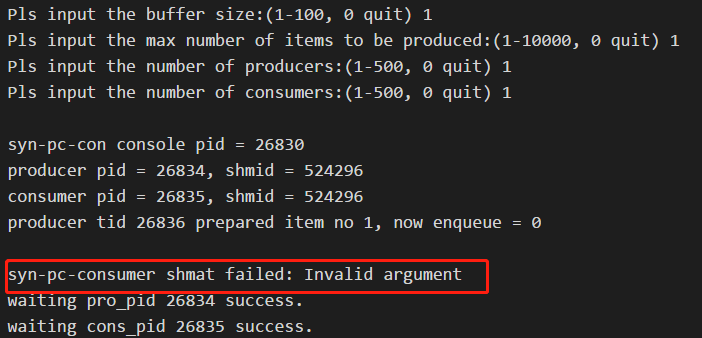


生产者先生产3个产品，此时缓冲区已满，消费者开始消费，条件1一直为真，直到消费了三个产品，此时条件1为假，逻辑覆盖完成。

条件2：if (shm == (void \*)-1)

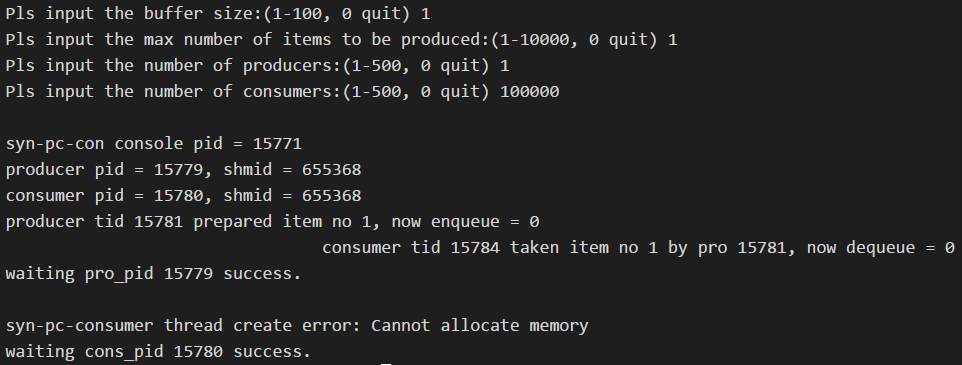


shmat为开启共享内存对该进程的访问，需要让该函数调用失败使条件2为True，修改传入的参数即可



条件3需要让子线程创建失败，因为在静态检查中提到的对输入参数处理的漏洞，所以可以实现这一点，测试用例如下：

|  |  |  |  |
| --- | --- | --- | --- |
| BUFFER\_SIZE | MAX\_ITEM\_NUM | THREAD\_PRO | THREAD\_CONS |
| 1 | 1 | 1 | 10000 |



可以看到打印了创建子线程错误的信息，说明条件3进入True分支，因为数量太多而导致内存不够，条件3已覆盖

条件4与条件2相同。

**黑盒测试**

a) 对主控模块 syn-pc-con-6.c 的输入实现等价类划分测试。

参数一共有四个，根据他们的取值范围设计测试用例

buffer\_size：1-100，0退出

max\_*item\_*num:：1-10000，0退出

producer：1-500，0退出

consumer：1-500，0退出

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | buffer\_size | max\_*item\_*num | thread\_pro | thread\_cons | 预期结果 |
| W1 | 0 | 1 | 1 | 1 | 输入0之后退出程序 |
| W2 | 1 | 0 | 1 | 1 |
| W3 | 1 | 1 | 0 | 1 |
| W4 | 1 | 1 | 1 | 0 |
| W5 | 101 | 1 | 1 | 1 | 输入超过范围的数据会提示重新输入 |
| W6 | 1 | 10001 | 1 | 1 |
| W7 | 1 | 1 | 501 | 1 |
| W8 | 1 | 1 | 1 | 501 |
| W9 | -1 | 1 | 1 | 1 |  |
| W10 | 1 | -1 | 1 | 1 |  |
| W11 | 1 | 1 | -1 | 1 |  |
| W12 | 1 | 1 | 1 | -1 |  |
| W13 | 5 | 3 | 1 | 1 |  |
| W14 | 3 | 5 | 1 | 1 |  |
| W15 | 5 | 3 | 3 | 3 |  |
| W16 | 3 | 5 | 3 | 3 |  |

W1234:

输入0之后退出程序，符合预期



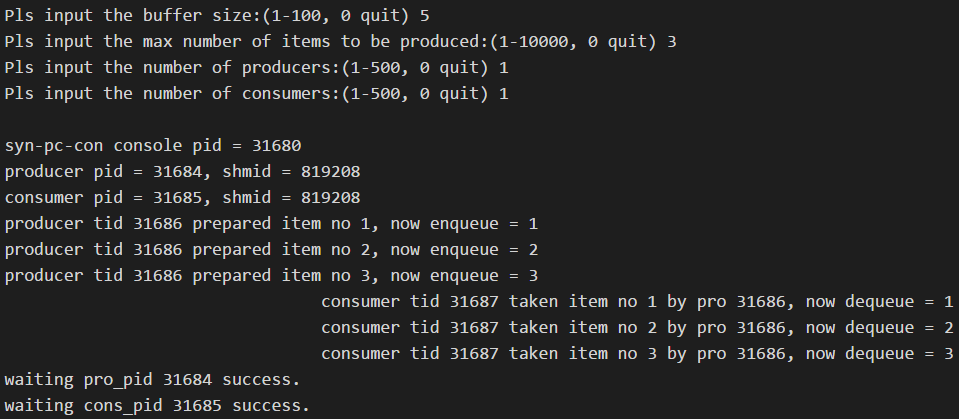
W5678:



W9,10,11,12:

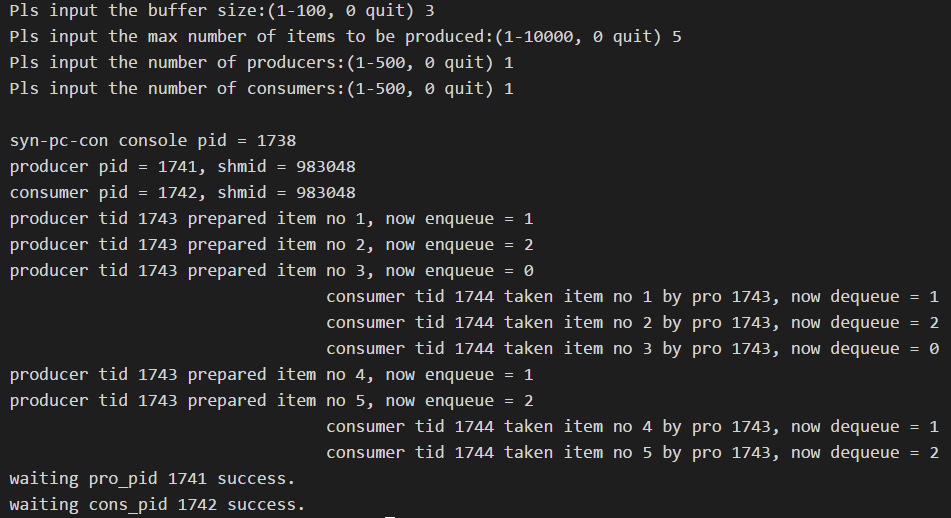


W13:



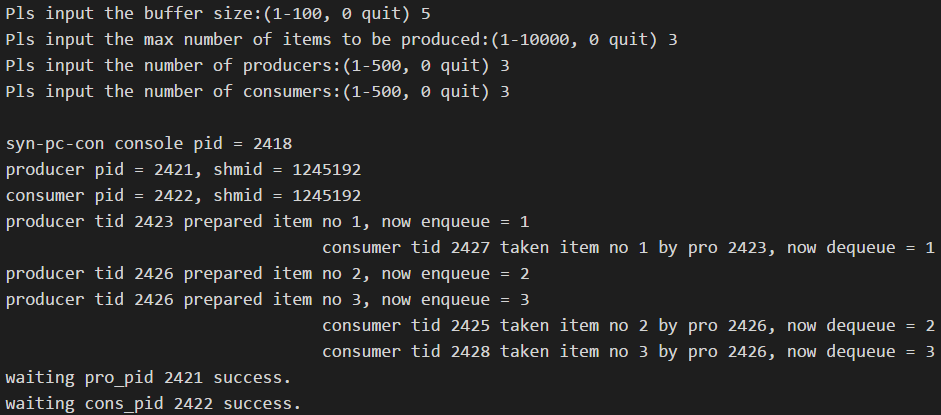
缓冲区比最大产品数量大，线程数都为1时，生产者线程会占用锁，，一次生产完所有产品，消费者再开始消费。

W14：



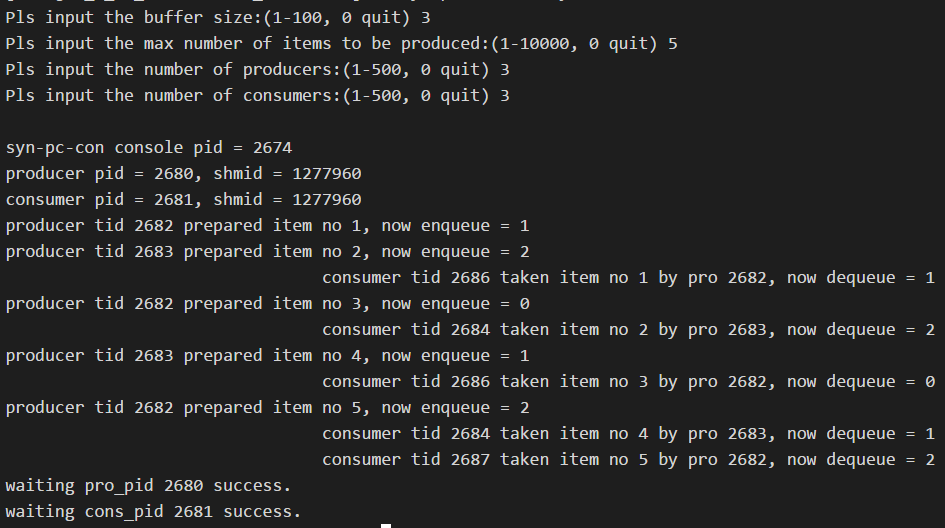
缓冲区比最大产品数小，线程数都为1，缓冲区满之后停止生产，消费者开始消费，由于都是单线程， 所以会一直消费完所有产品再由生产者生产完剩余的产品，然后再次消费。

W15：



缓冲区比最大产品数量大，线程数都为3， 线程的调度结果不可预测。

W16：



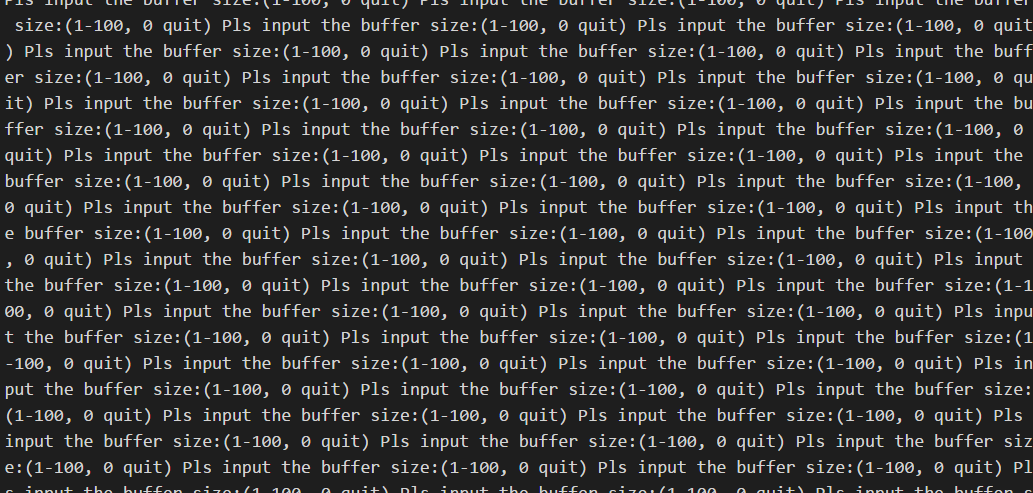
缓冲区比最大产品数量小，线程数都为3， 线程的调度结果不可预测。

**系统测试**

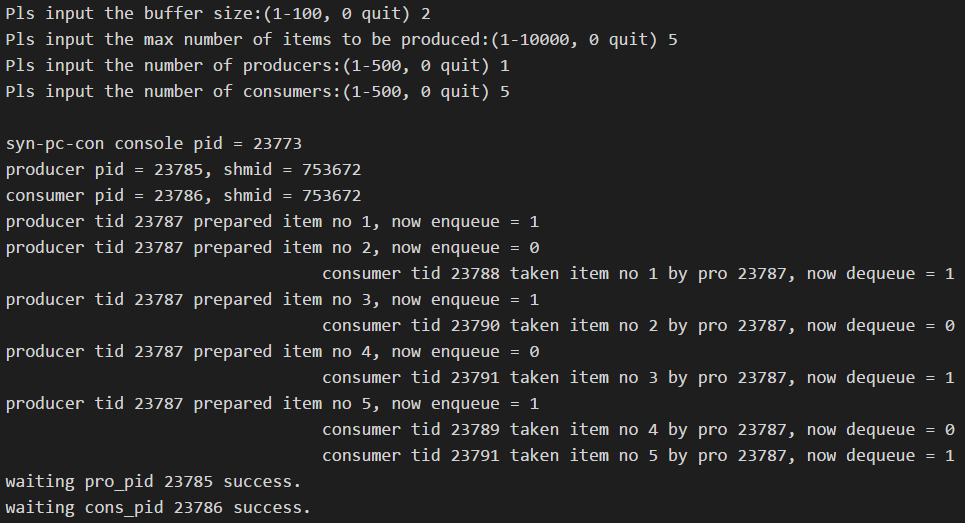
a) 自行选择两种故障模型进行软件故障静态注入测试。

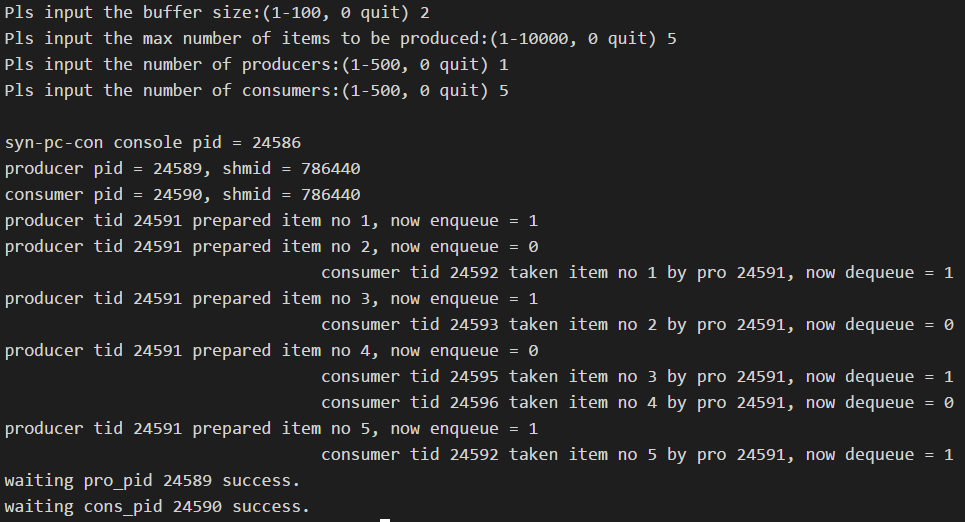
1. 输入非法数据

输入字符串，会一直死循环，并未对字符串做异常处理。



2. 同样输入产生不同结果



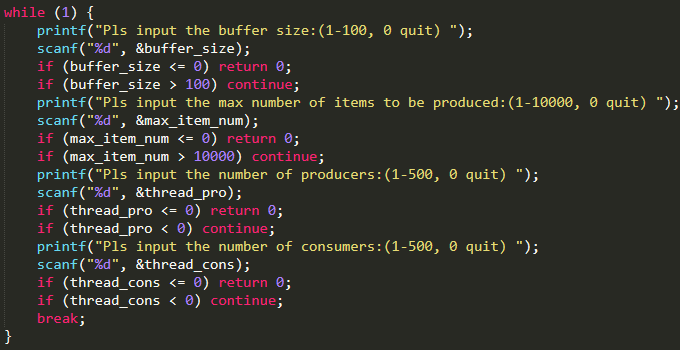


因为线程调度是随机的，所以可能会出现同一个用例线程调度的顺序不一样的情况。

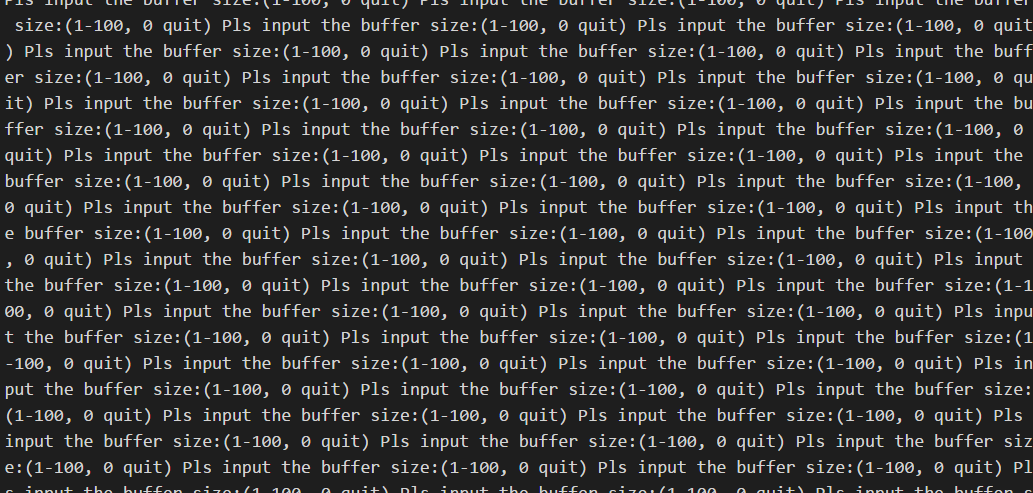
**【测试结果】**

**一共发现缺陷：5处**

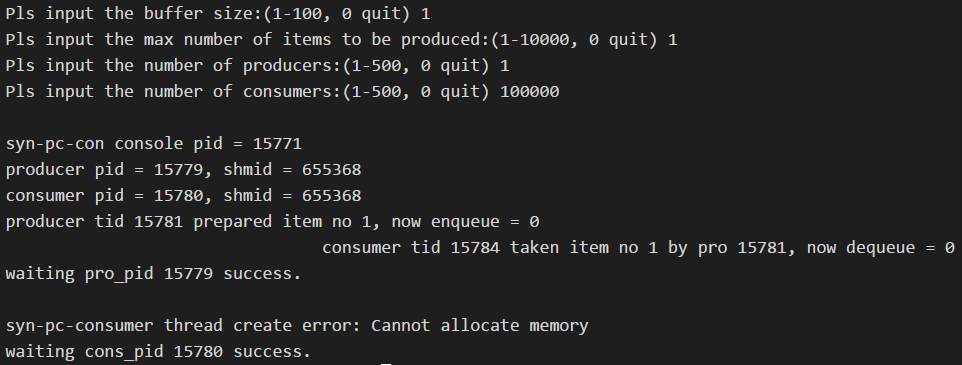
1. **头文件包含错误**
2. **无法进入的分支路径**



1. **任何一个参数输入字符串死循环**



1. **线程数输入10000没有判断超过边界，并且会出现内存错误**



1. **功能说明中只有运行方法，并没有具体编译方法，可能出现用户编译时文件名写错导致无法运行程序的情况。**

**【技术日志】**

**奇怪的 (void \*)-1**

shm = shmat(shmid, 0, 0);  
if (shm == (void \*)-1) {  
 perror("\nsyn-pc-consumer shmat failed");  
 exit(EXIT\_FAILURE);  
}

(void \*)-1表示把 -1 转换为无类型指针 0xFFFFFFF，以前没有见过这样的条件判断，所以了解了一下用途。

shmat函数的原型为:

void \*shmat(int shm\_id, const void \*shm\_addr, int shcmflg);

调用成功时返回一个指向共享内存第一个字节的指针，如果调用失败返回-1，类型为void \*，所以用shm == (void \*)-1来判断shmat是否调用成功。