**实验三：同步问题**

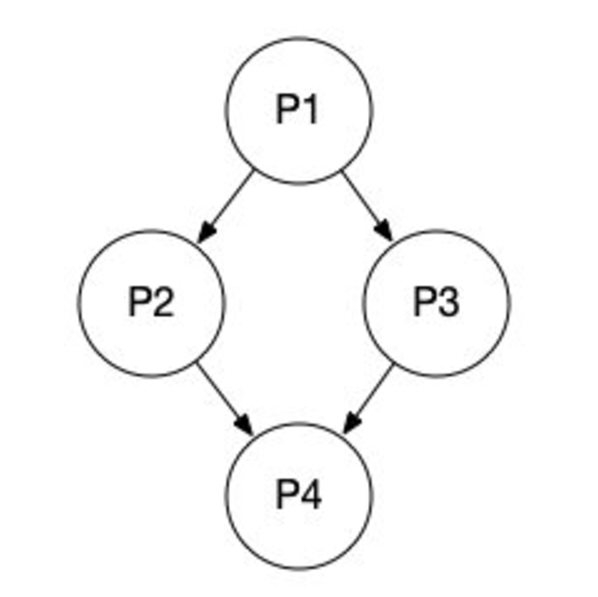
**安全1601 16281221 邓子轩**

1. **通过fork的方式，产生4个进程P1,P2,P3,P4，每个进程打印输出自己的名字，例如P1输出“I am the process P1”。要求P1最先执行，P2、P3互斥执行，P4最后执行。通过多次测试验证实现是否正确。**

代码如下：



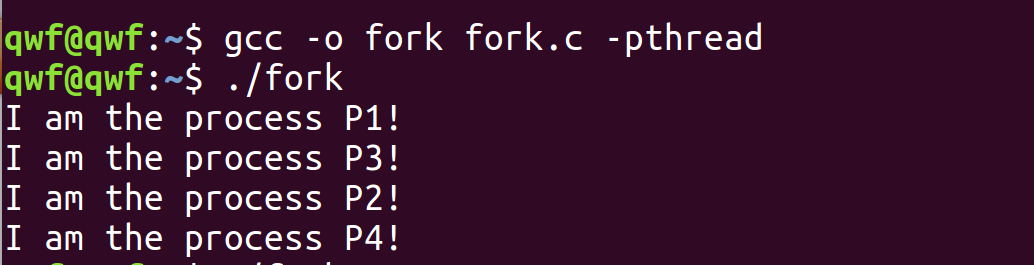
按照要求建立的的四个进程的前趋图如下：



在代码中通过信号量P23\_signal来实现P2与P3的互斥关系，当P1执行完毕后为P23\_signal增加一个可用资源，之后P2跟P3竞争，取得信号量的进程将消耗掉这个资源，执行完毕后则释放一个资源，则另一个进程得以执行。

P2\_signal、P3\_signal初值为0，当P2、P3均执行完毕后，二者资源值才均为1。P4进程通过wait操作等待两个进程（P2、P3）的结束。

运行结果如下：



1. **火车票余票数ticketCount 初始值为1000，有一个售票线程，一个退票线程，各循环执行多次。添加同步机制，使得结果始终正确。要求多次测试添加同步机制前后的实验效果。(说明：为了更容易产生并发错误，可以在适当的位置增加一些pthread\_yield()，放弃CPU，并强制线程频繁切换，例如售票线程的关键代码：**

**temp=ticketCount;**

**pthread\_yield();**

**temp=temp-1;**

**pthread\_yield();**

**ticketCount=temp;**

**退票线程的关键代码：**

**temp=ticketCount;**

**pthread\_yield();**

**temp=temp+1;**

**pthread\_yield();**

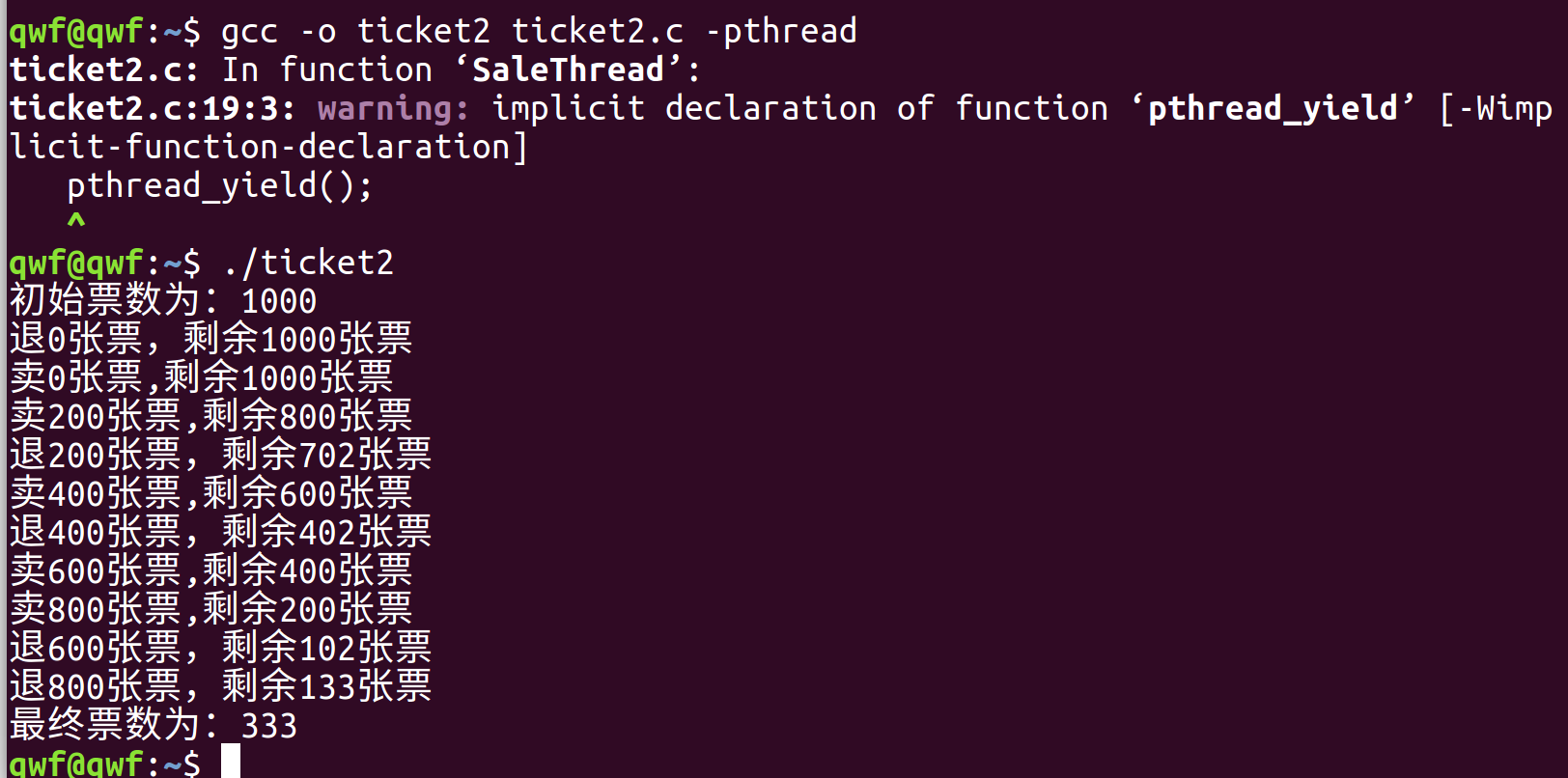
**ticketCount=temp;**

程序如下：

****

****

运行结果如下：

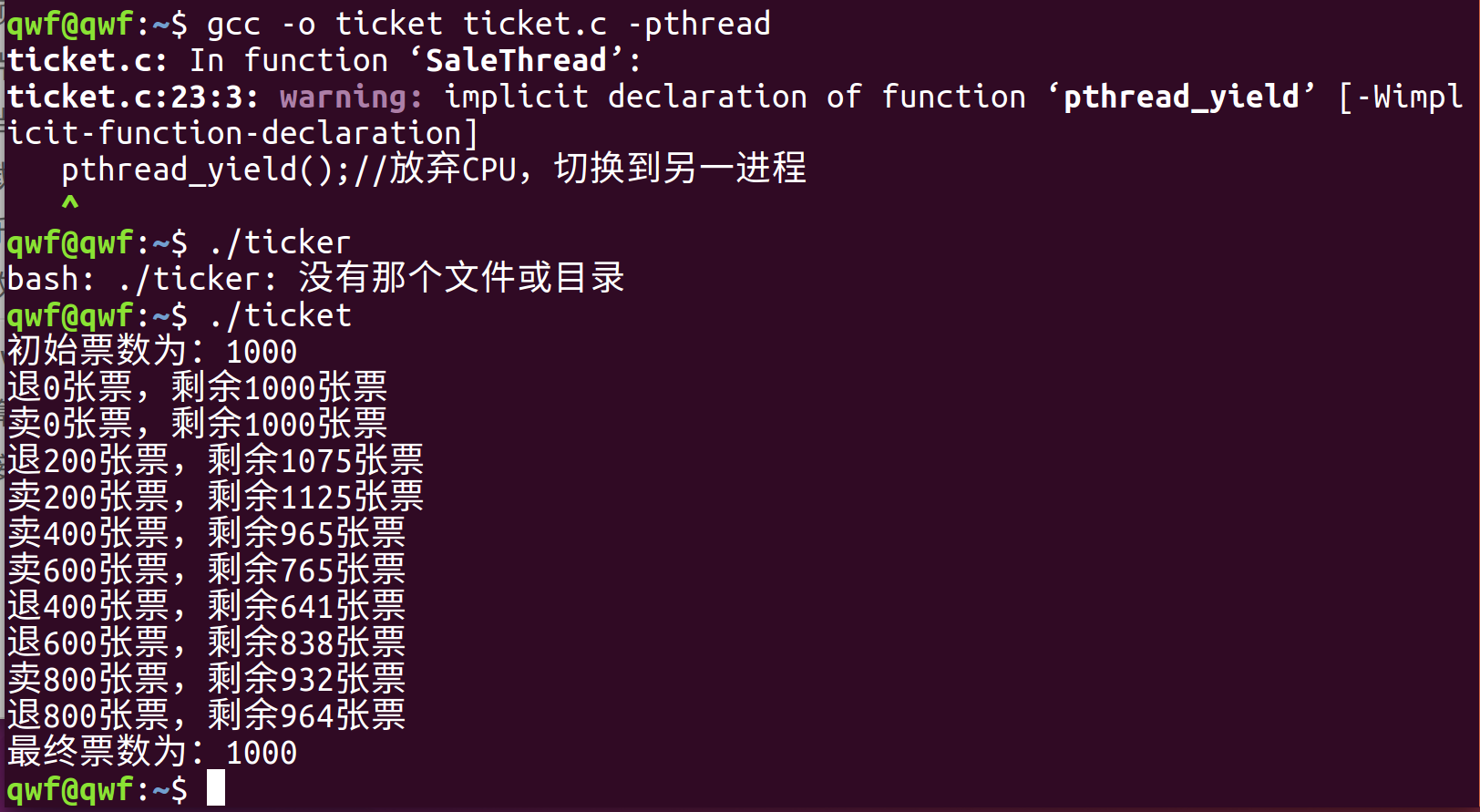
****

可以看出因为无同步机制时同时访问变量可能会发生不可重复读等问题，最终票数几乎不可能正确。

加入同步机制后程序如下：



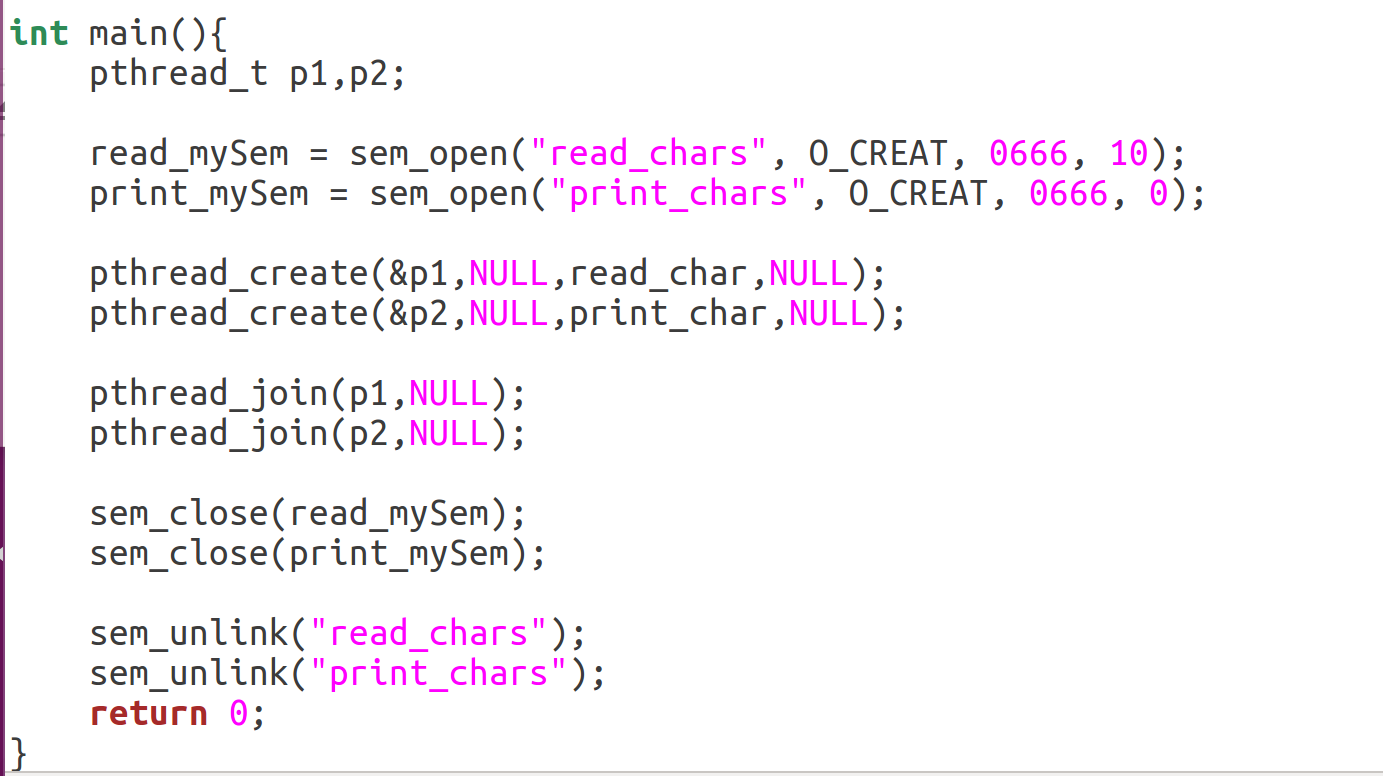
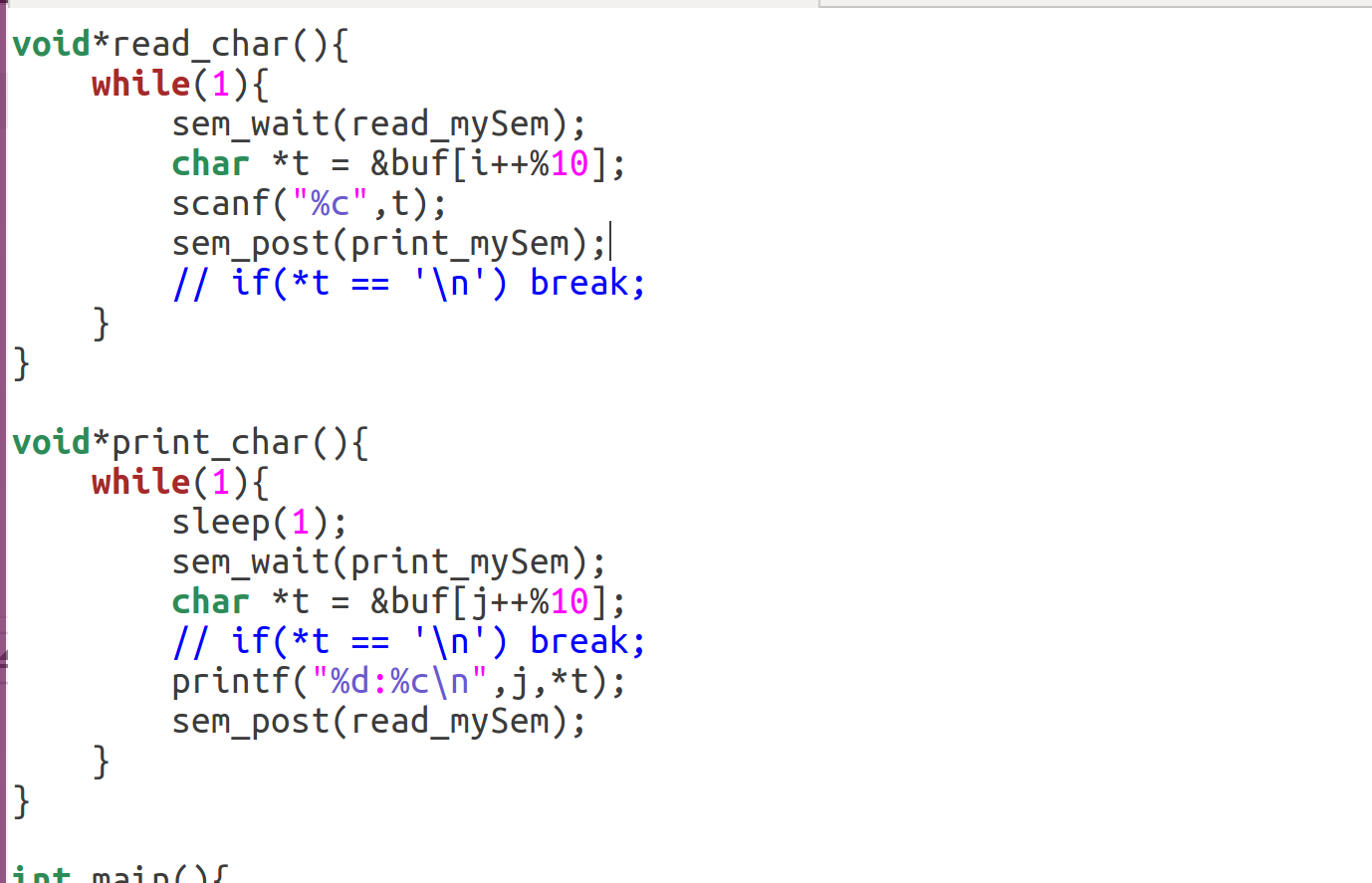
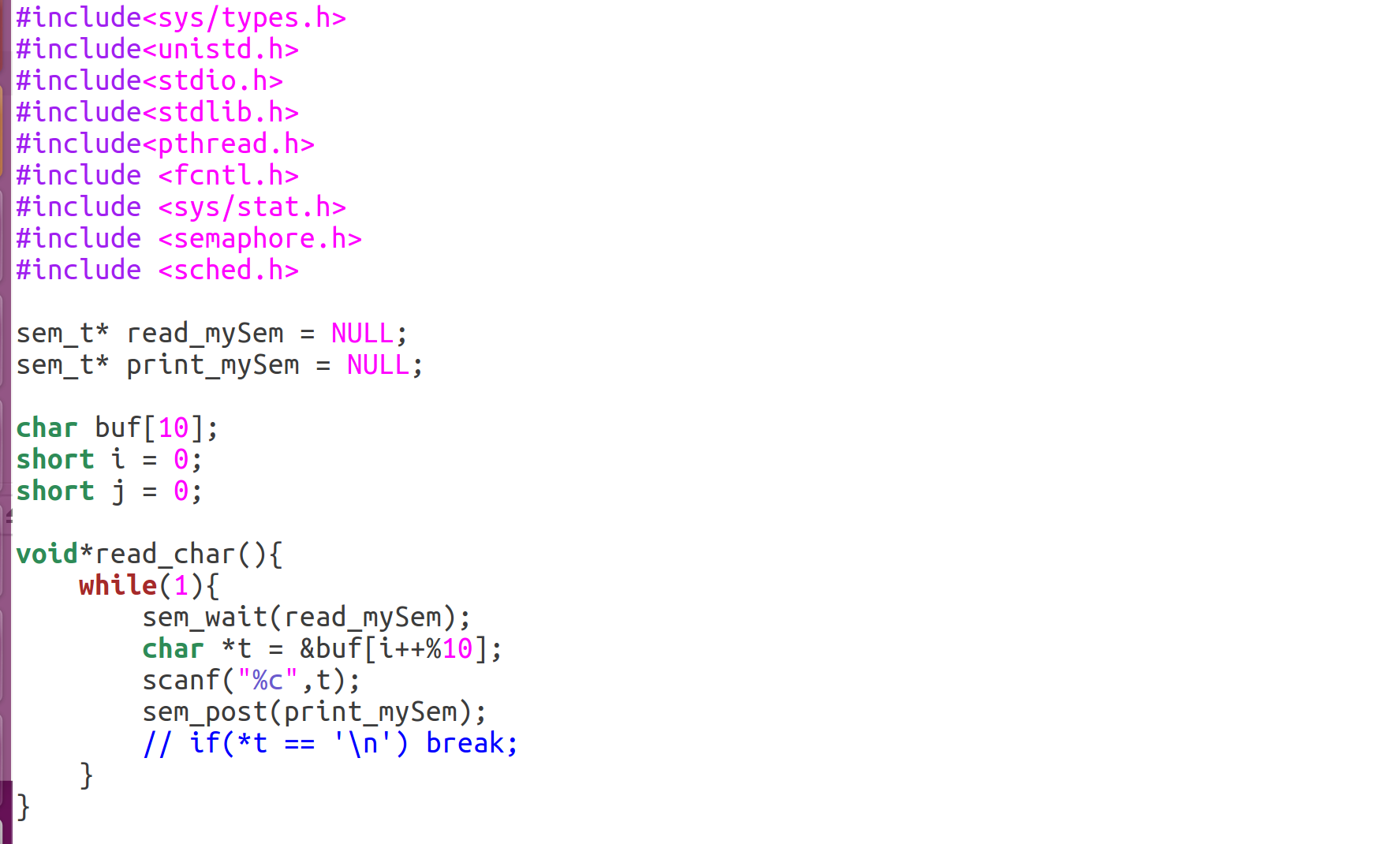
运行结果如下：



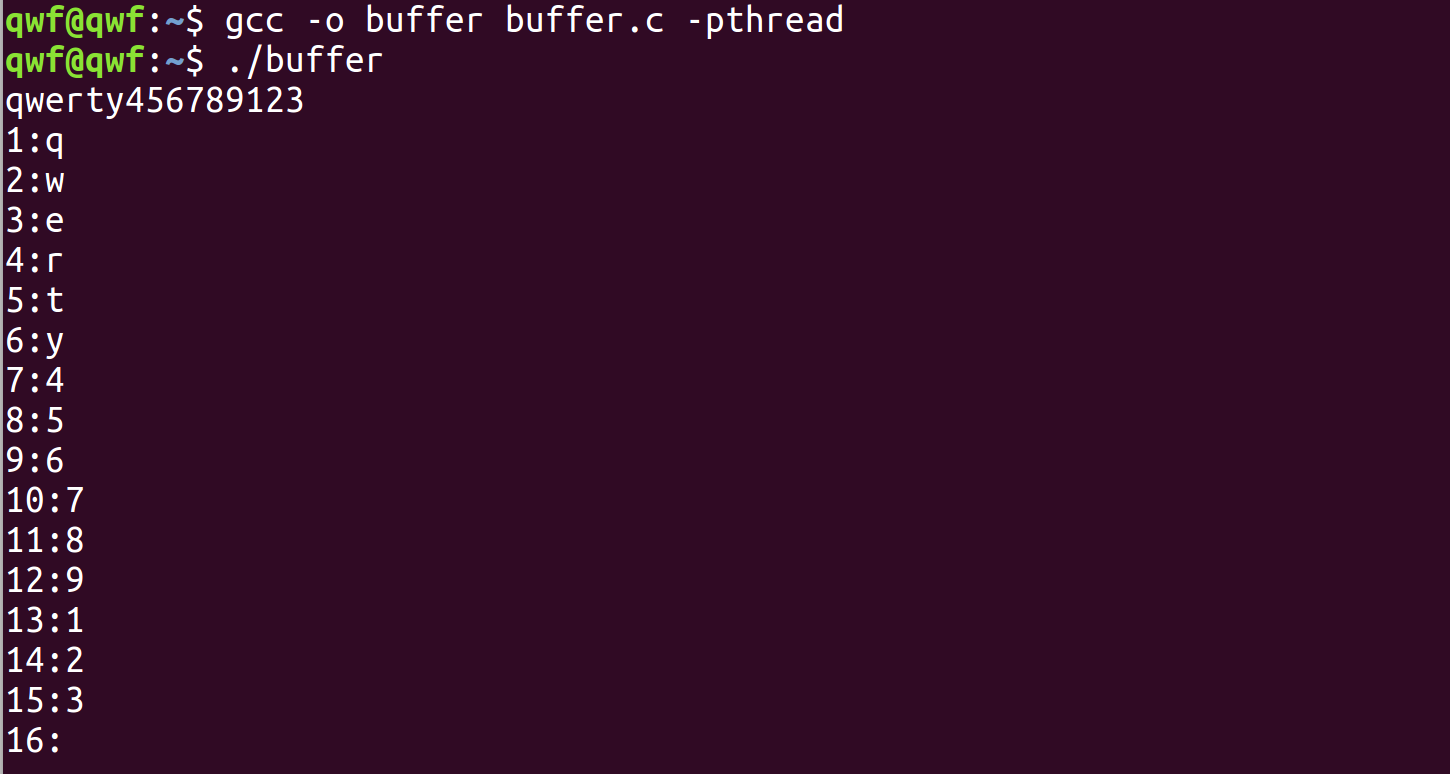
可以看到经过各1000次加减和频繁的进程调度，最终结果依然正确。

1. **一个生产者一个消费者线程同步。设置一个线程共享的缓冲区， char buf[10]。一个线程不断从键盘输入字符到buf,一个线程不断的把buf的内容输出到显示器。要求输出的和输入的字符和顺序完全一致。（在输出线程中，每次输出睡眠一秒钟，然后以不同的速度输入测试输出是否正确）。要求多次测试添加同步机制前后的实验效果。**

代码如下：



运行结果如下：



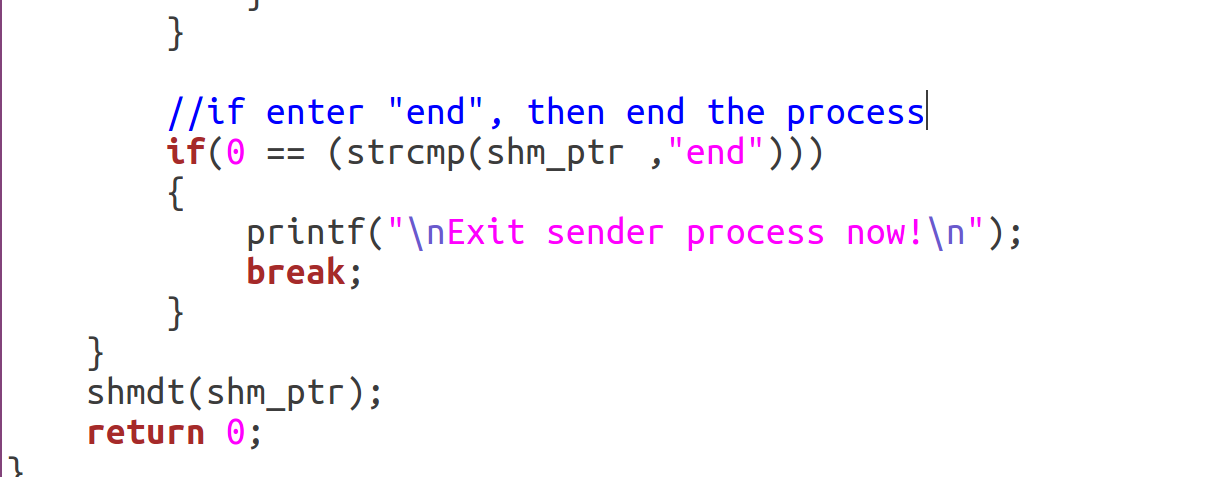
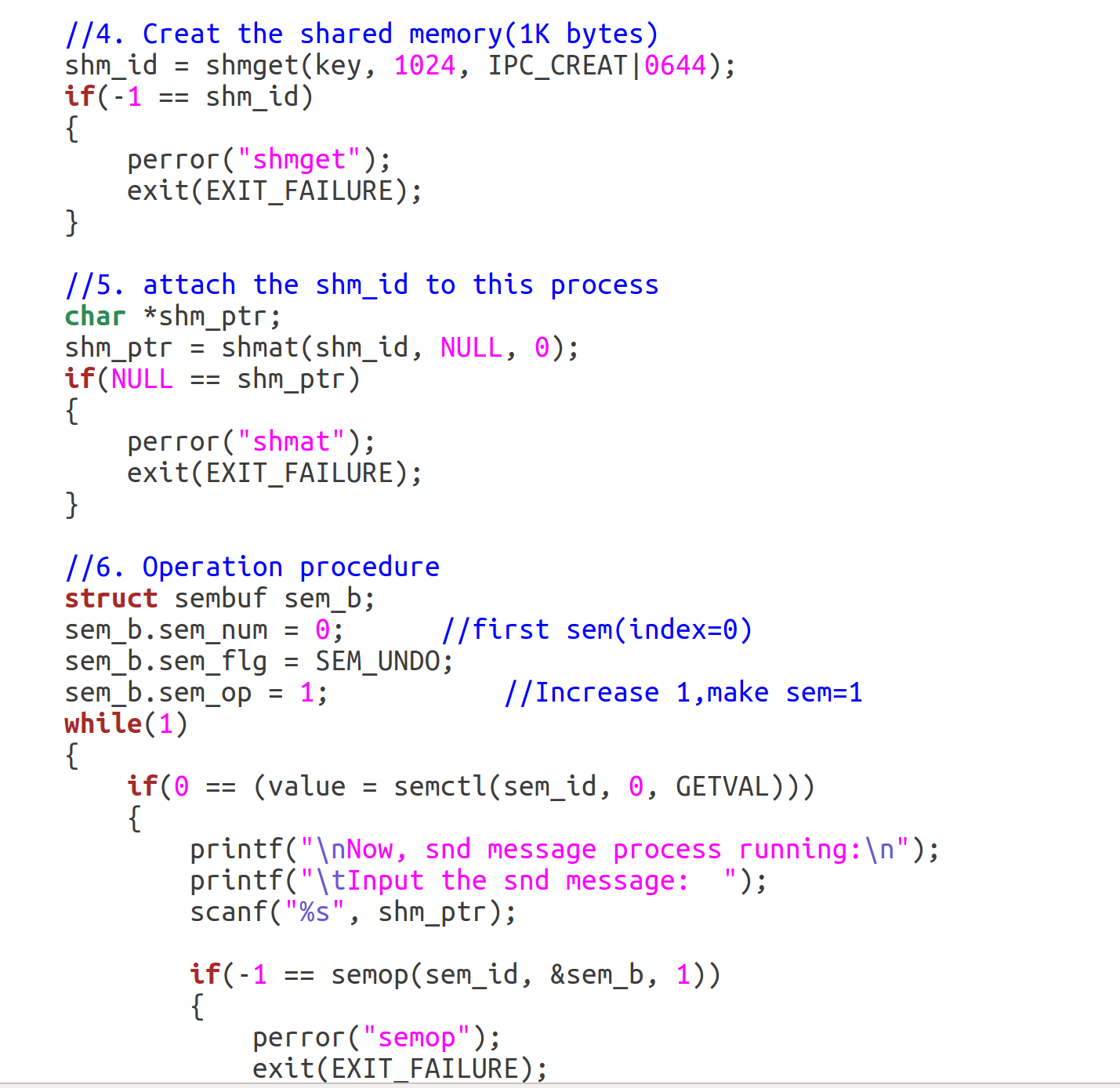
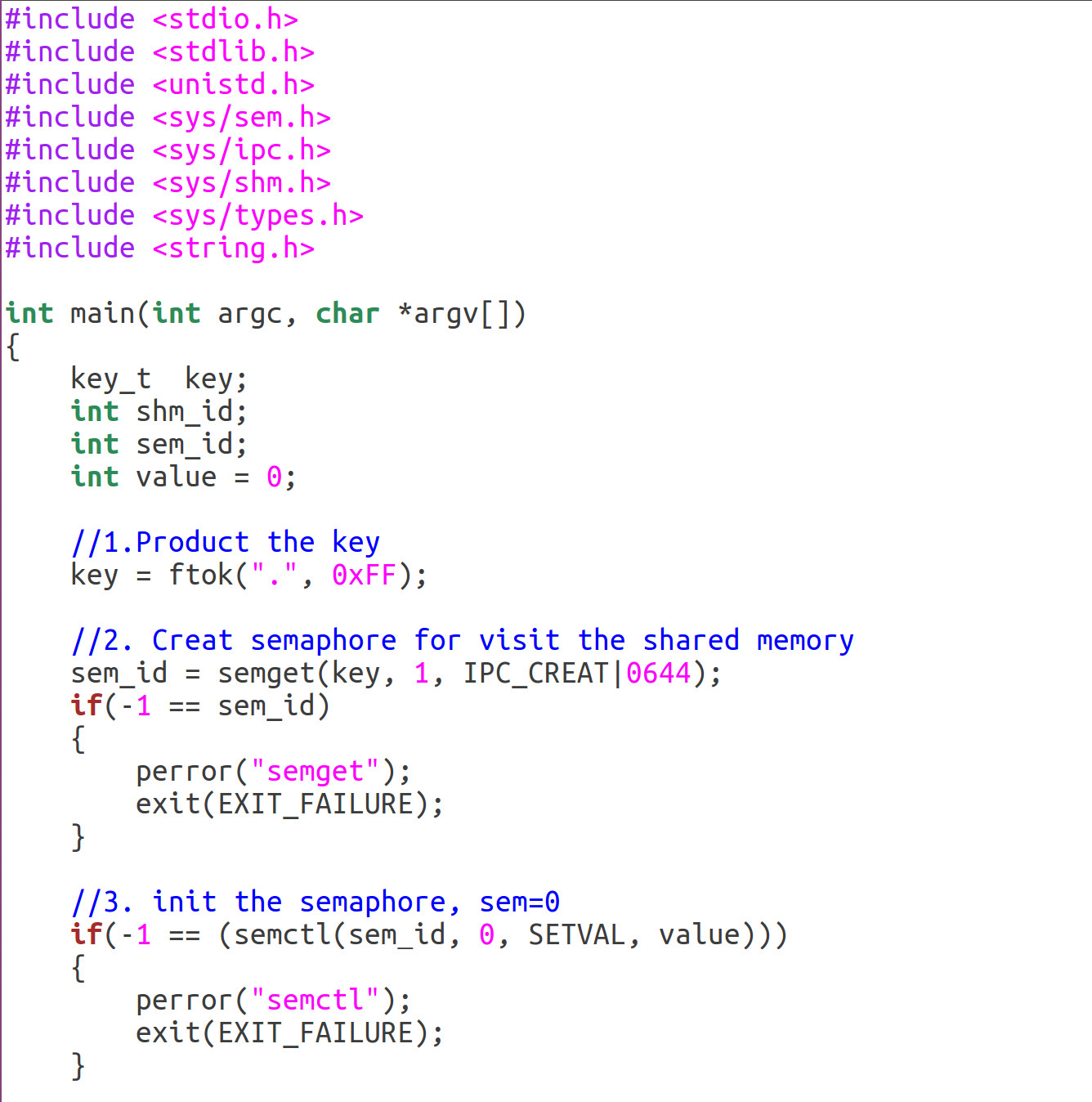
字符缓冲区共有10个位置，并且两个进程读取和打印的最小单位即字符。所以，等效初始状态有10个资源。读进程每读取一个字符存入缓冲区，缓冲区的资源数就会减少，以保证不会出现覆盖性写入的问题。打印进程输出的前提是，字符缓冲区有了新的字符输入，以免重复输出已经输出的字符。所以初始的输出信号量是0，每次读入一个字符后会加1。

**4.进程通信问题。阅读并运行共享内存、管道、消息队列三种机制的代码**

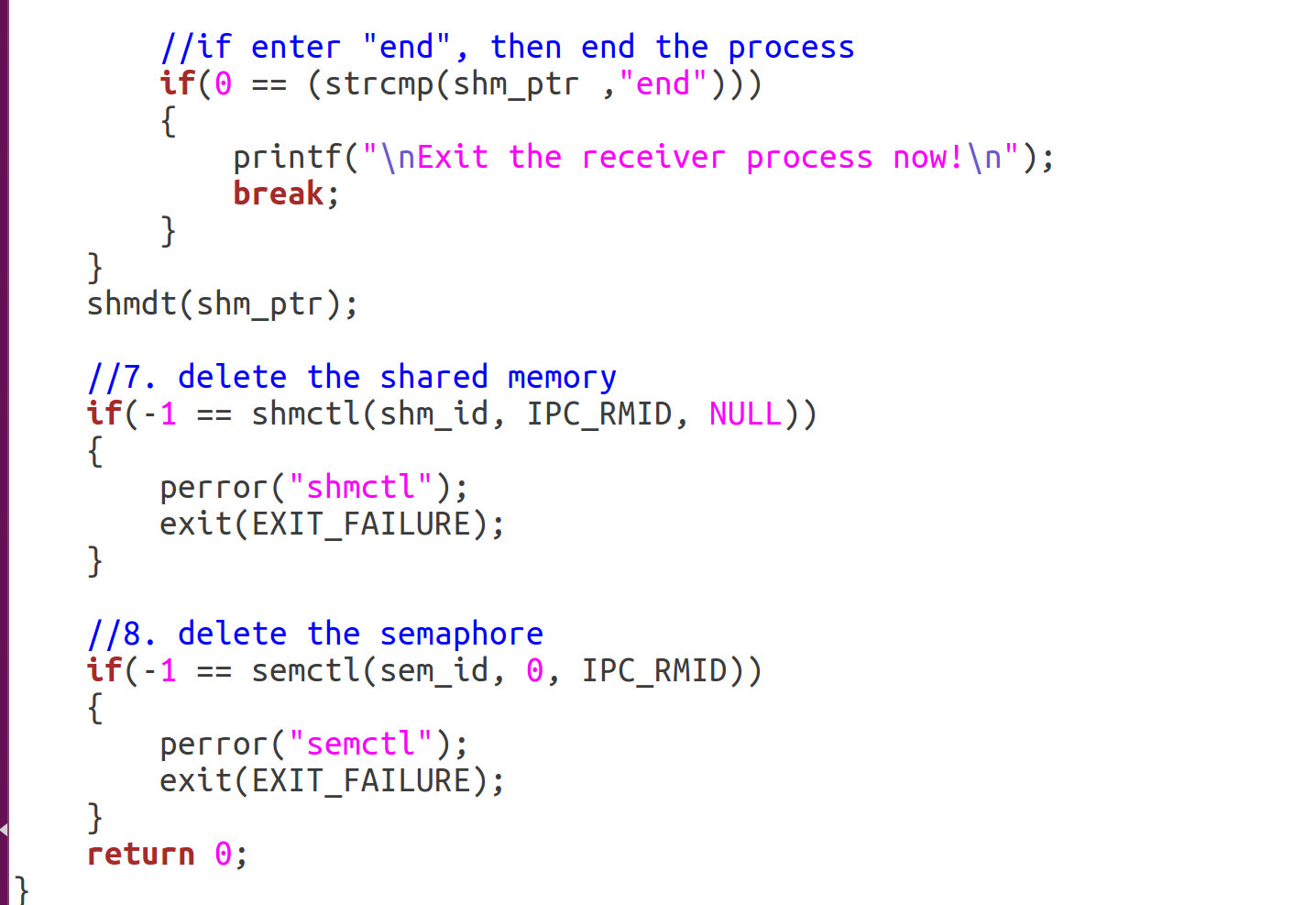
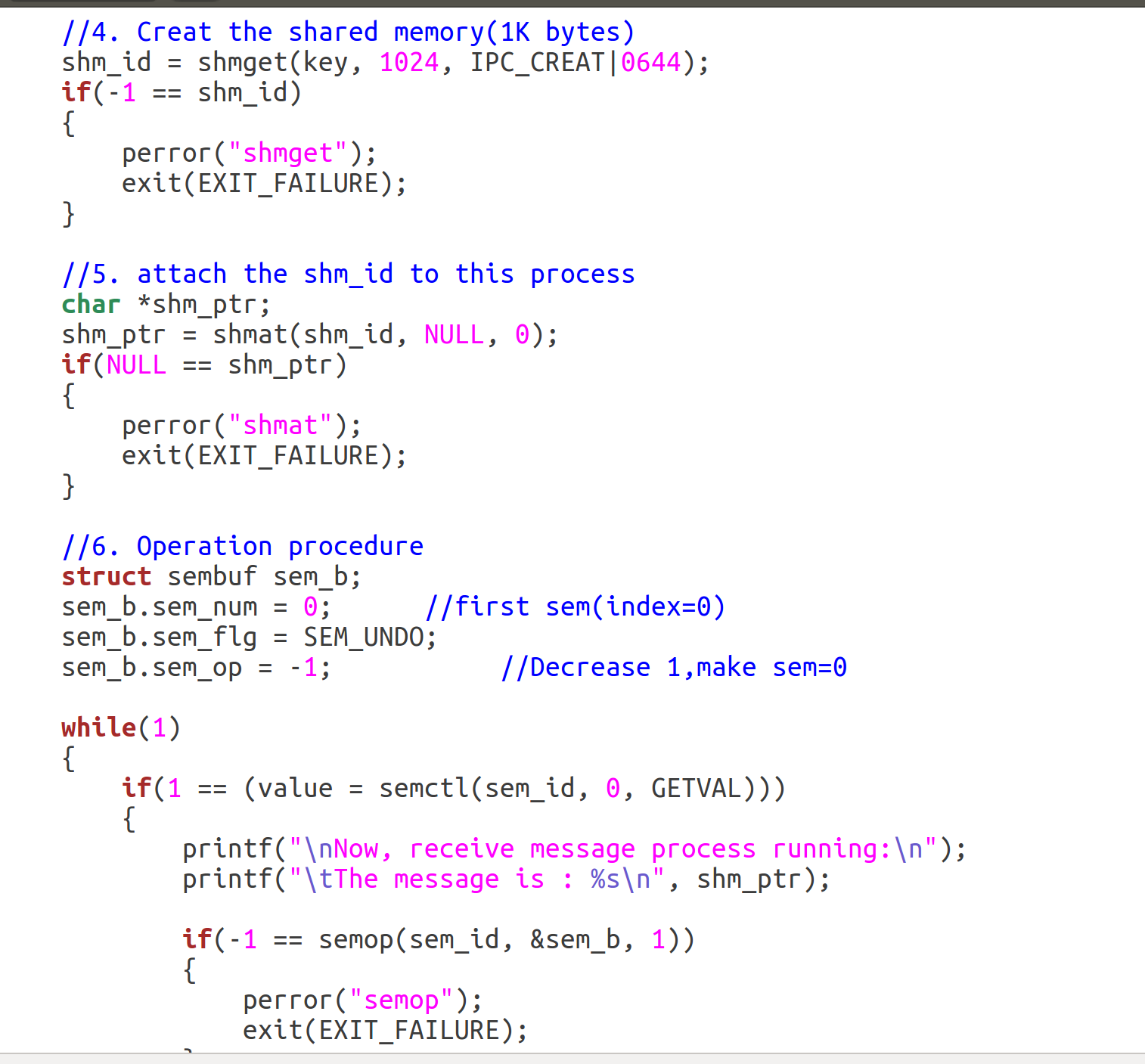
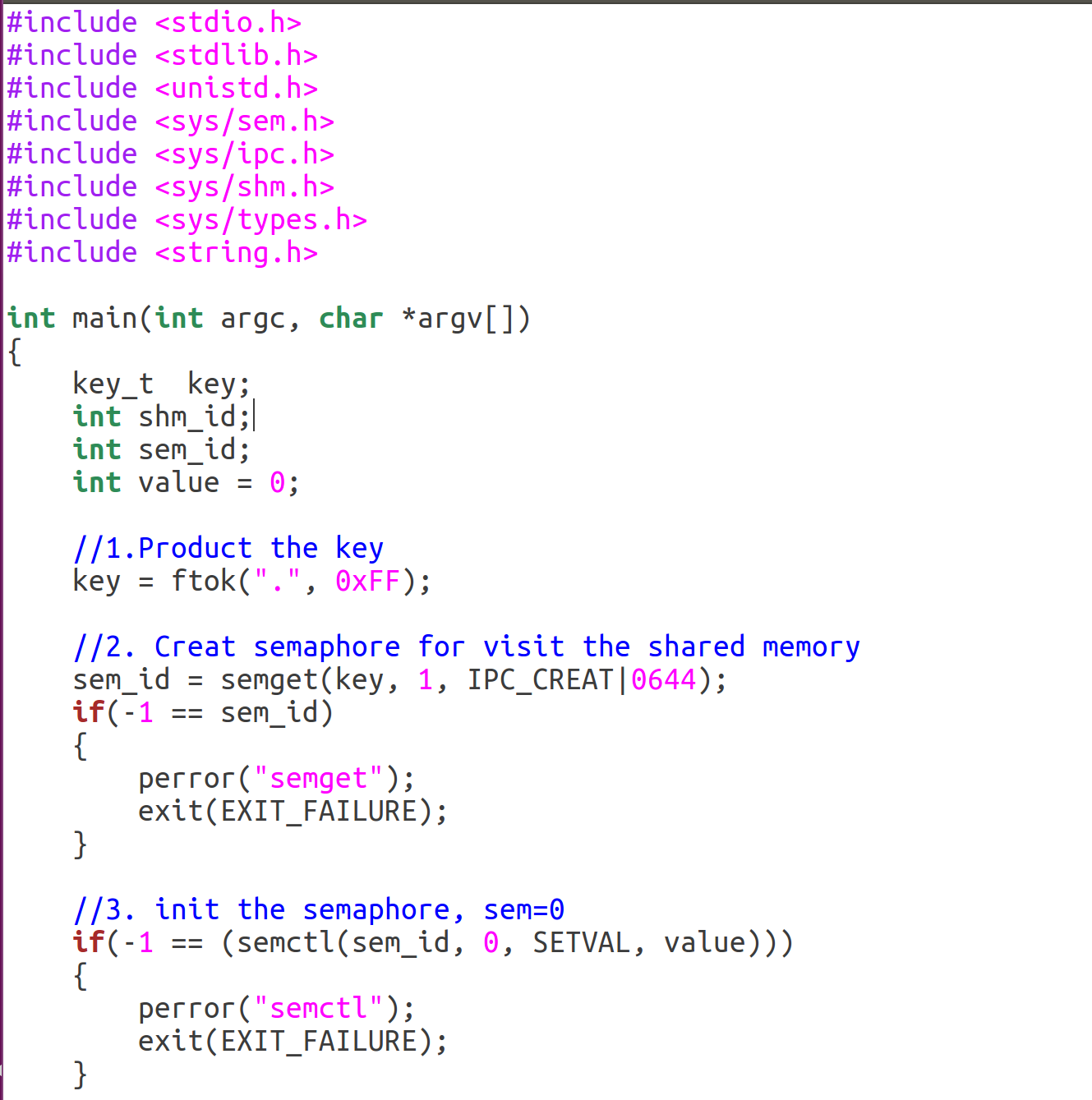
**实验测试**

**a）通过实验测试，验证共享内存的代码中，receiver能否正确读出sender发送的字符串？如果把其中互斥的代码删除，观察实验结果有何不同？如果在发送和接收进程中打印输出共享内存地址，他们是否相同，为什么？**

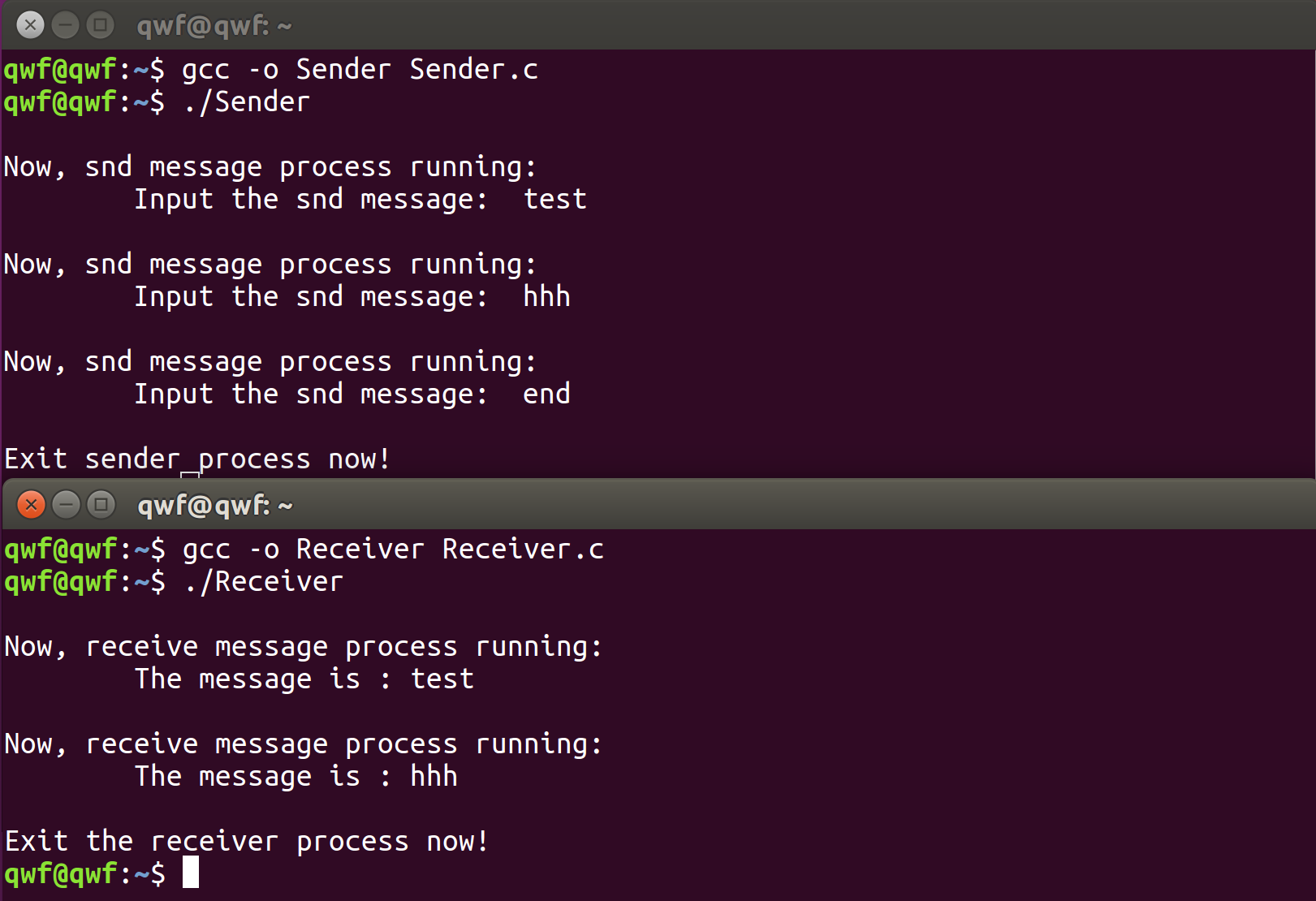
Sender.c:



Receiver.c：

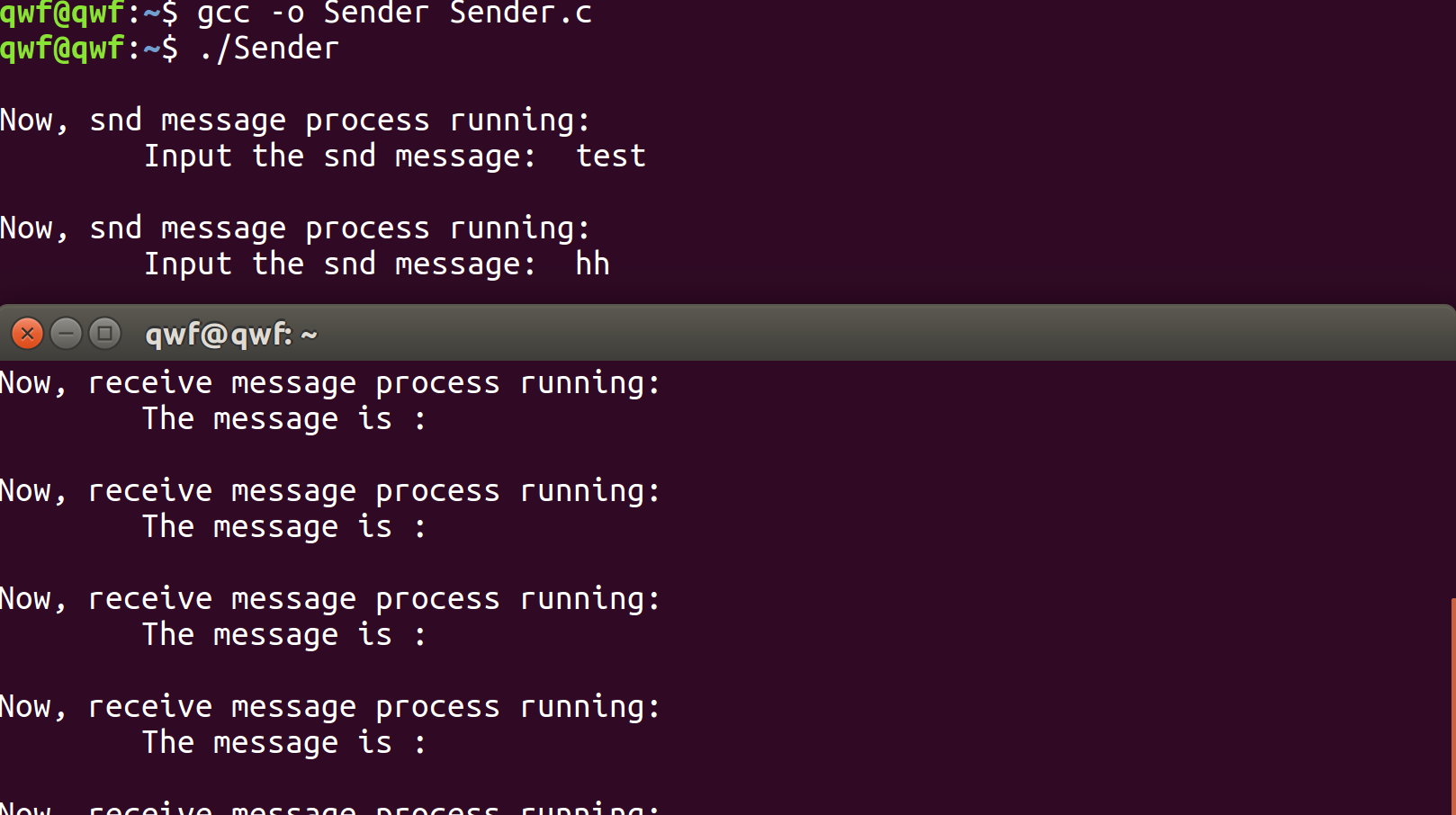


**运行结果如下：**



可以看到sender进程发出的消息receiver进程均准确无误的收到。

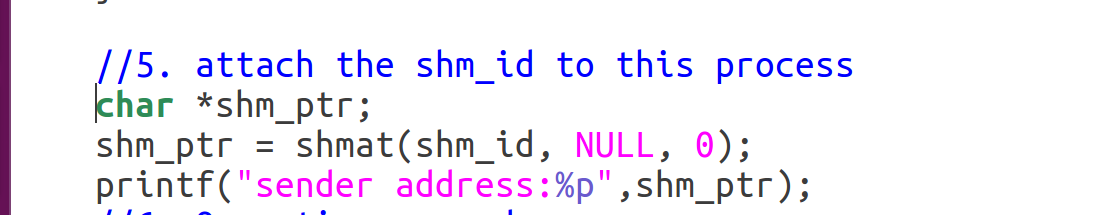
1.删除互斥访问相关的代码，运行结果如下：



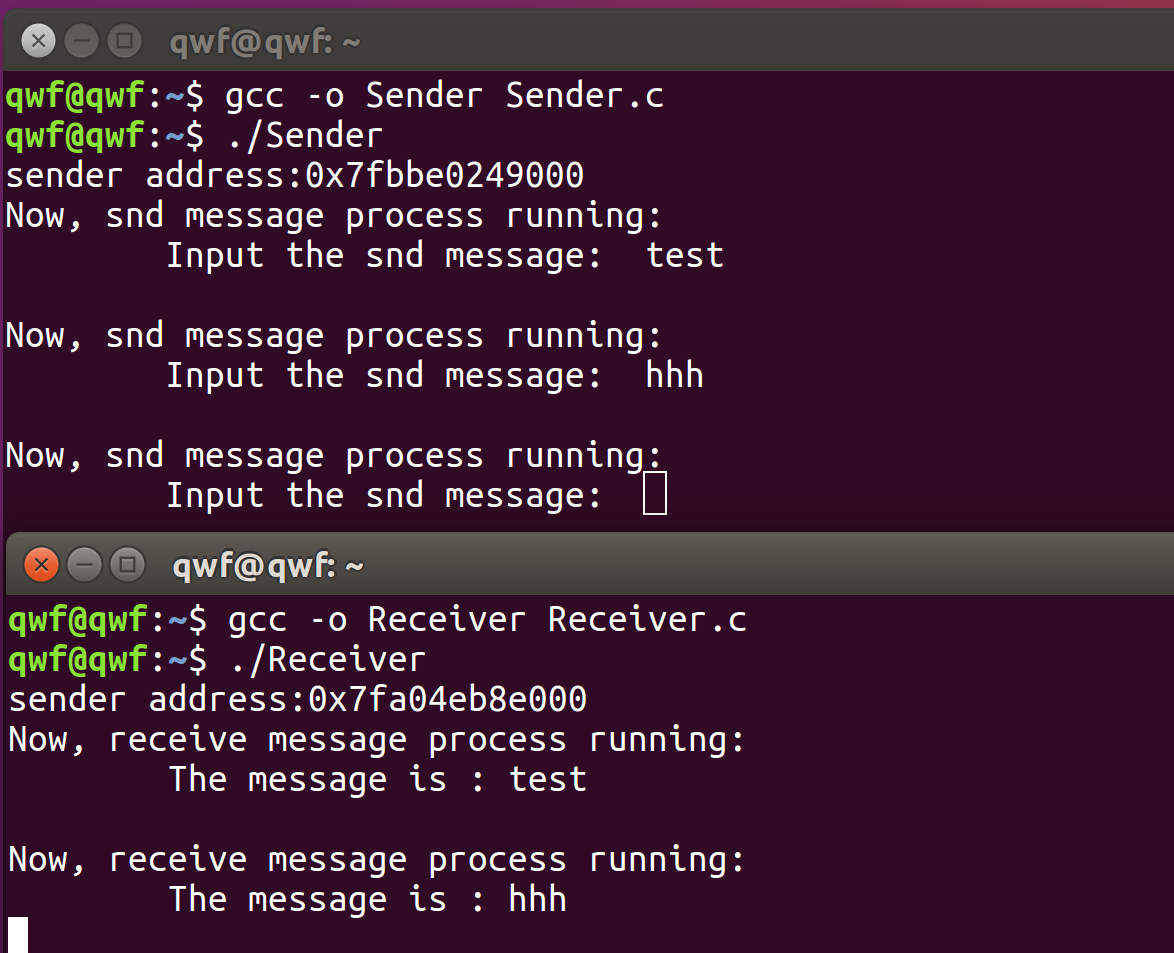
当删除互斥访问之后，两个进程便没有限制的访问共享内存，Sender进程由于受限于用户输入的速度，会停留一直等待用户输入数据，但是Receiver进程会一直输出共享内存中的消息。

2.打印共享地址：

修改如下：



运行结果如下



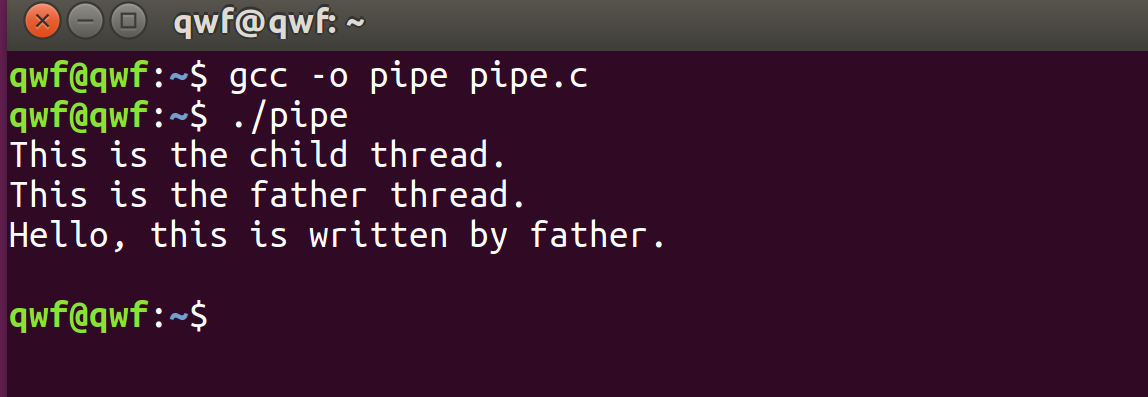
通过上面的现象可以看到共享内存在不同进程中是不相同的， 这是因为现代操作系统中都存在ASLR(地址空间随机化)，ASLR是⼀种针对缓冲区溢出的安全保护机制，具有ASLR机制的操作系统每次加载到内存的程序起始地址会随机变化。系统的这个随机化操作可能导致共享内存的地址不一致。

**b)有名管道和无名管道通信系统调用是否已经实现了同步机制？通过实验验证，发送者和接收者如何同步的。比如，在什么情况下，发送者会阻塞，什么情况下，接收者会阻塞？**

无名管道：



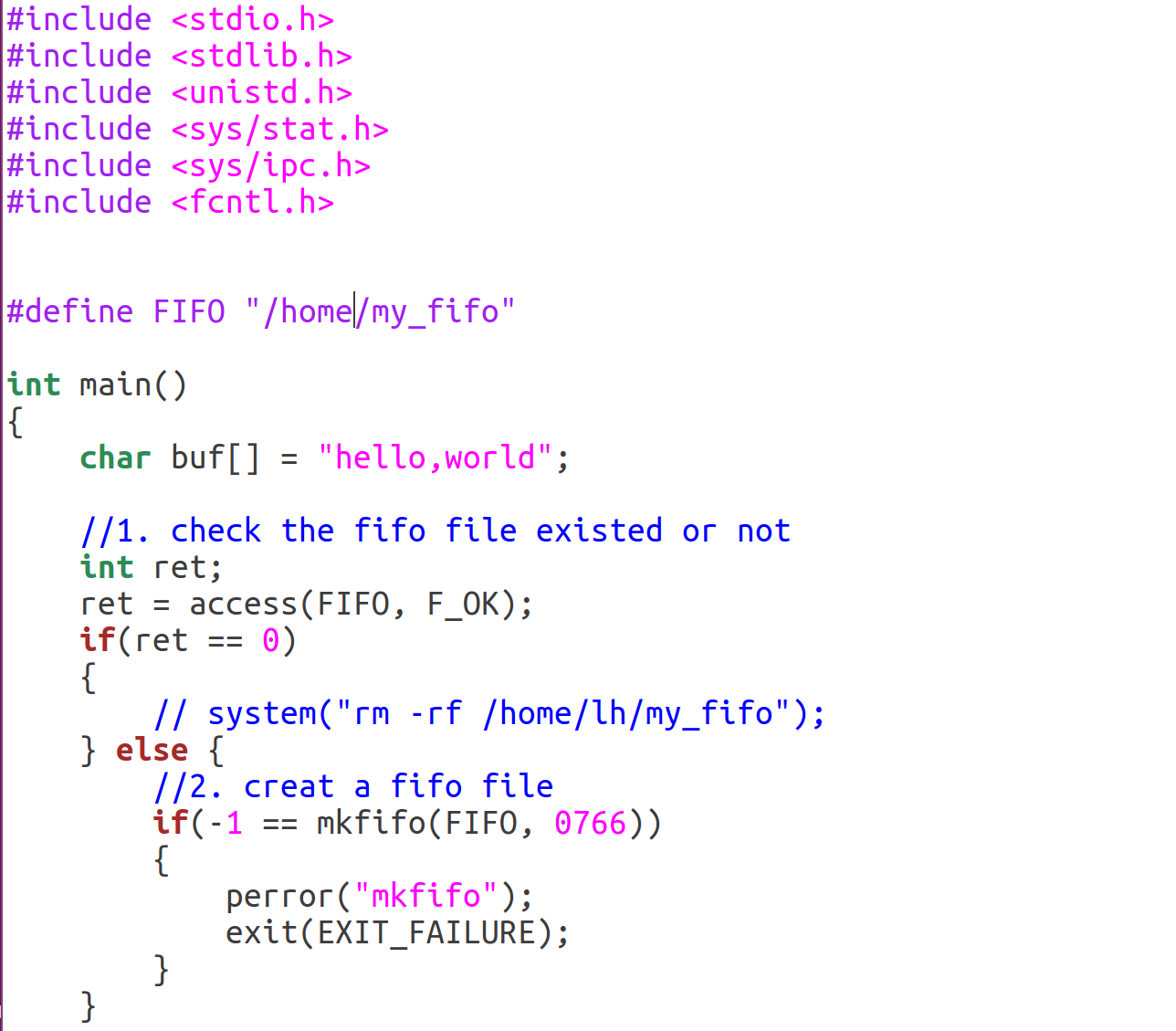
运行结果如下：



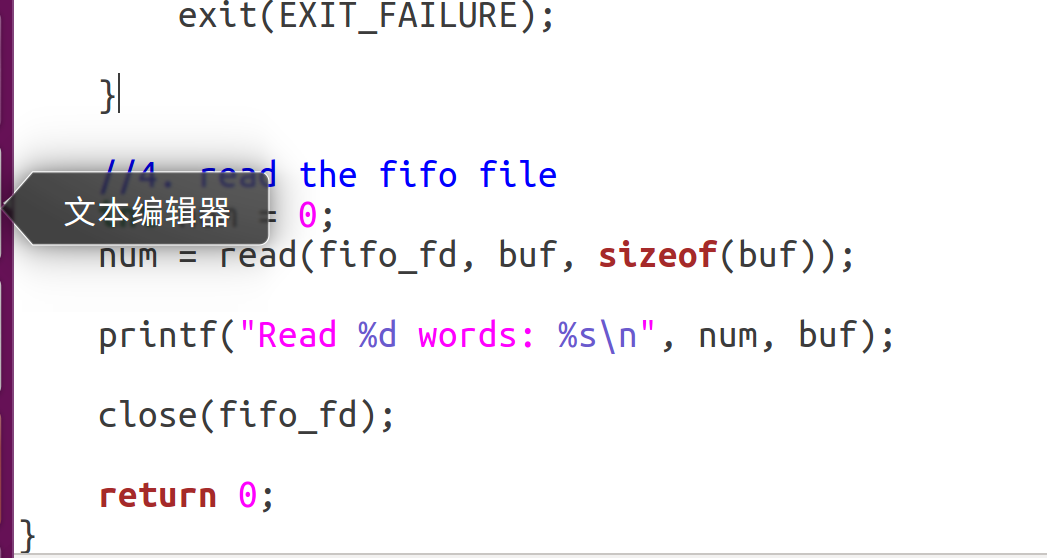
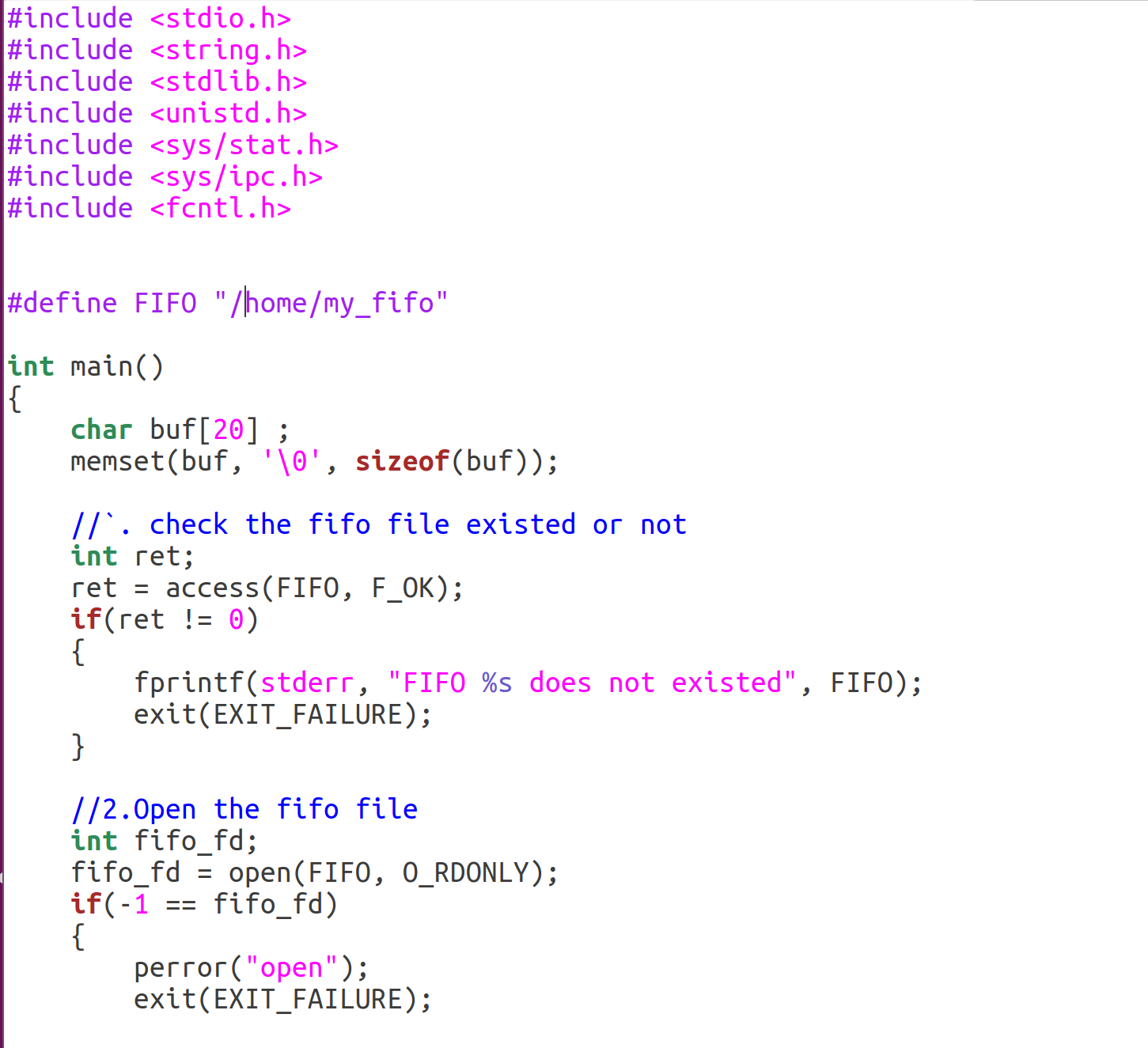
可以发现，在父进程未向管道写入数据前，子进程在原地阻塞；当父进程执行完毕写入操作时，子进程才重新获得CPU得以将管道内容输出。

有名管道：

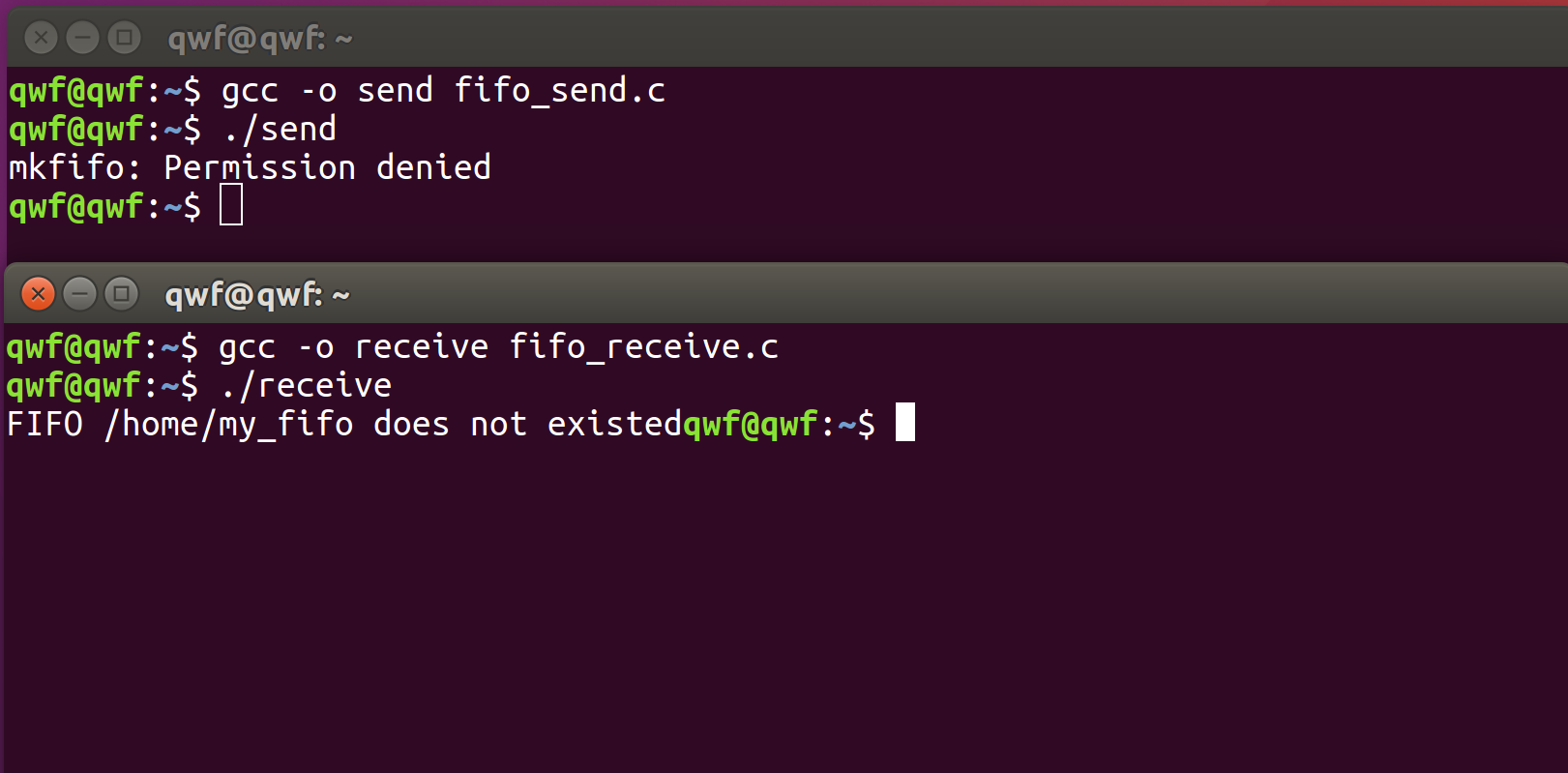
fifo\_send.c:



fifo\_receive.c:

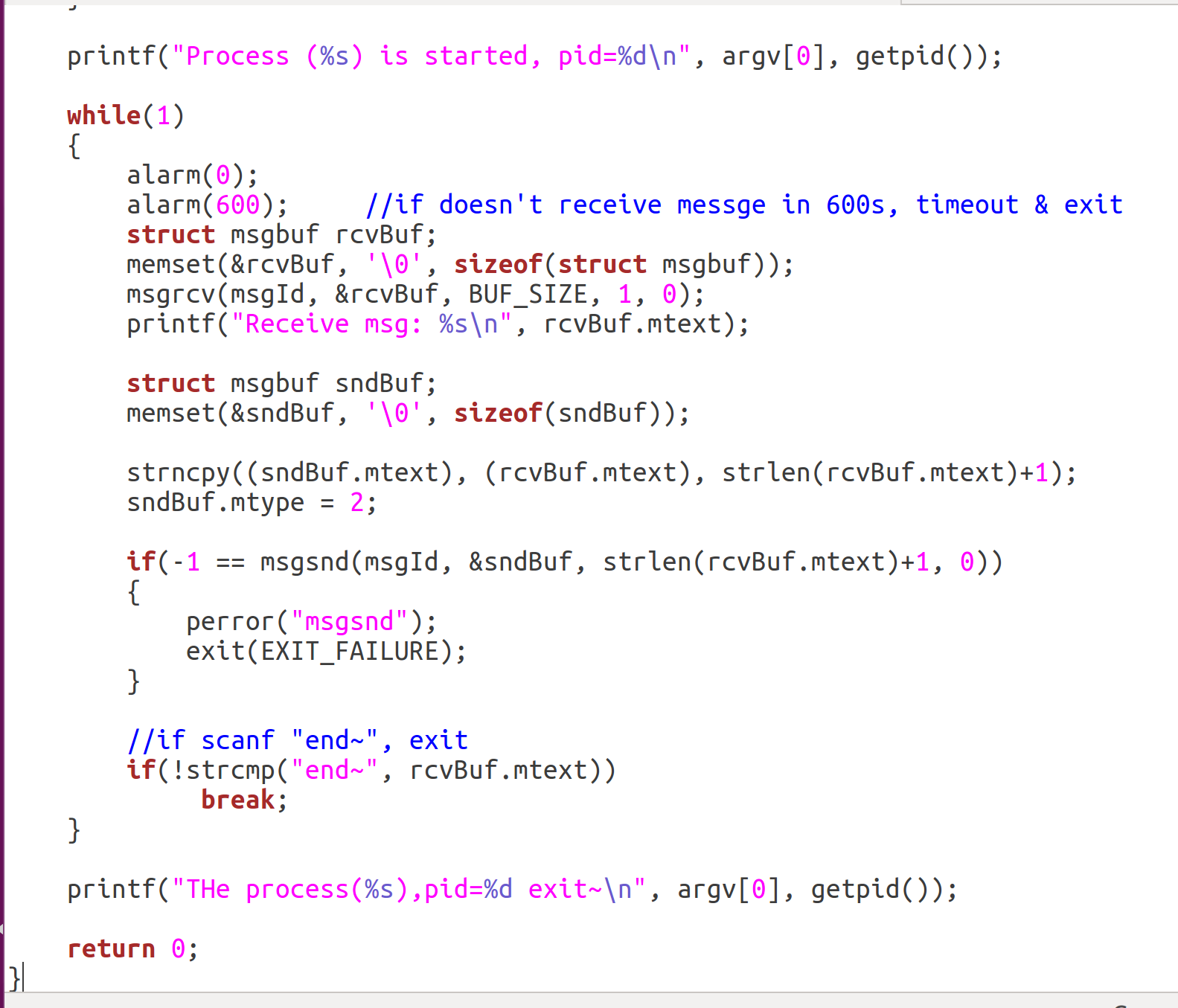
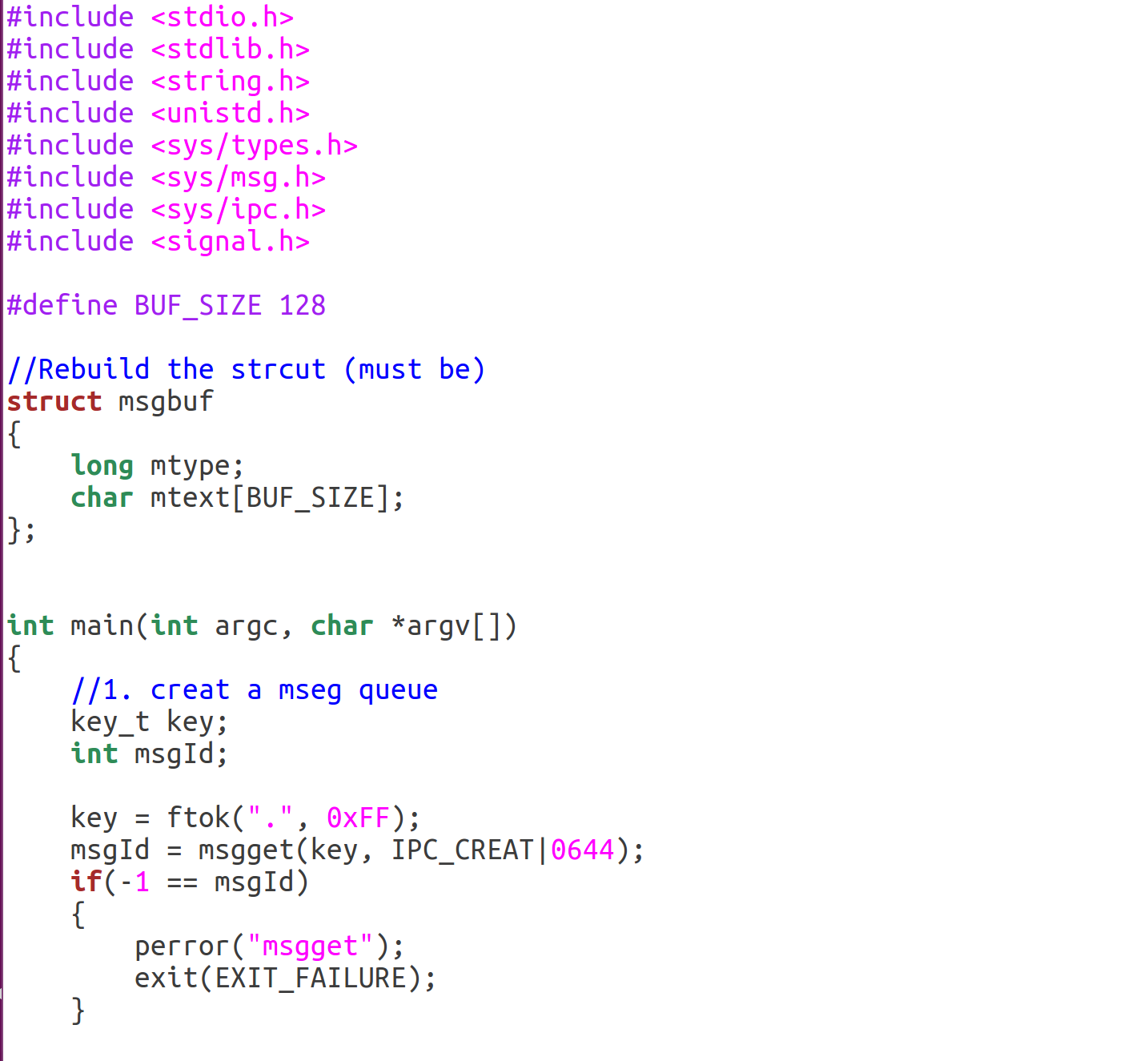


此实验运行失败：

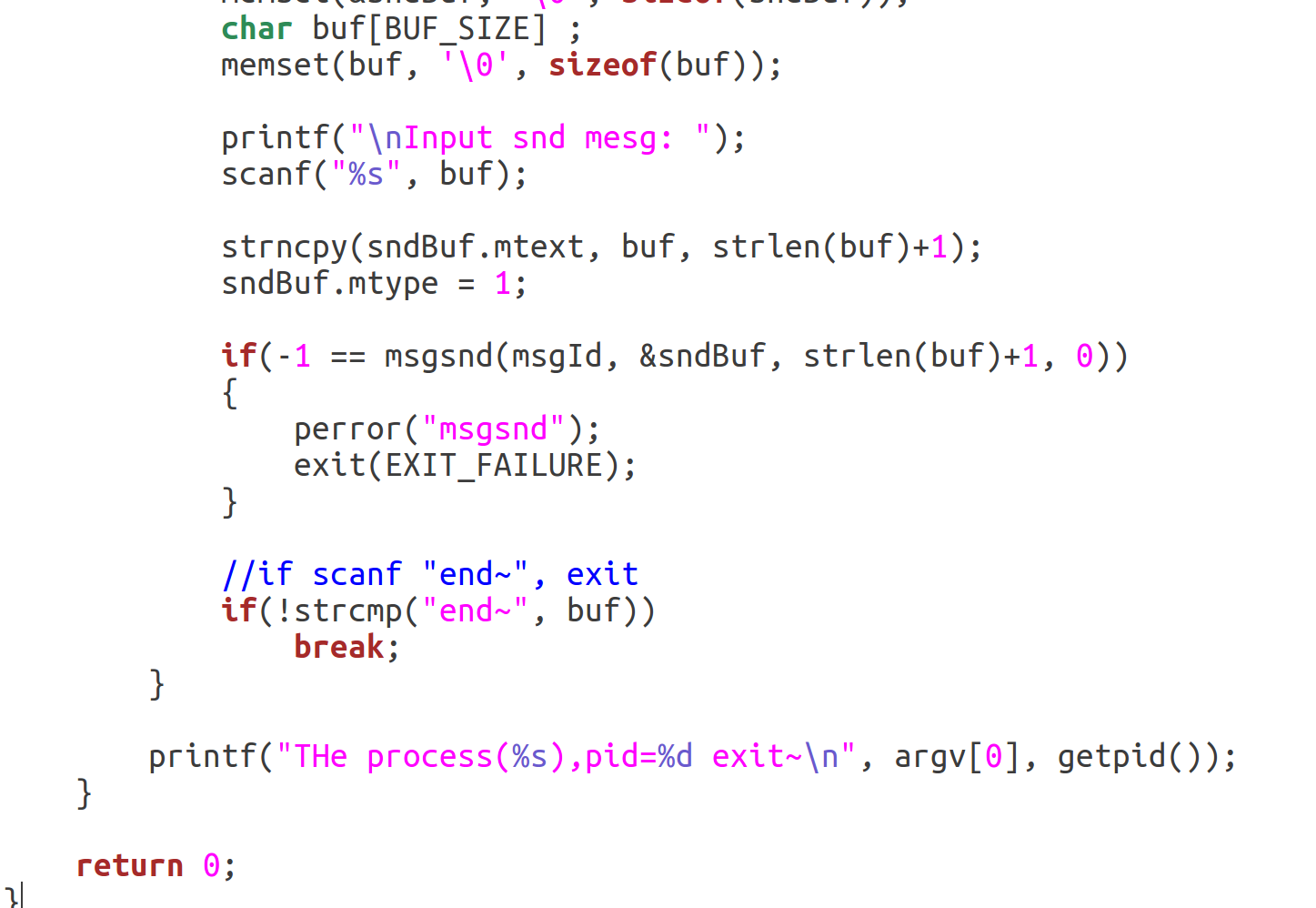
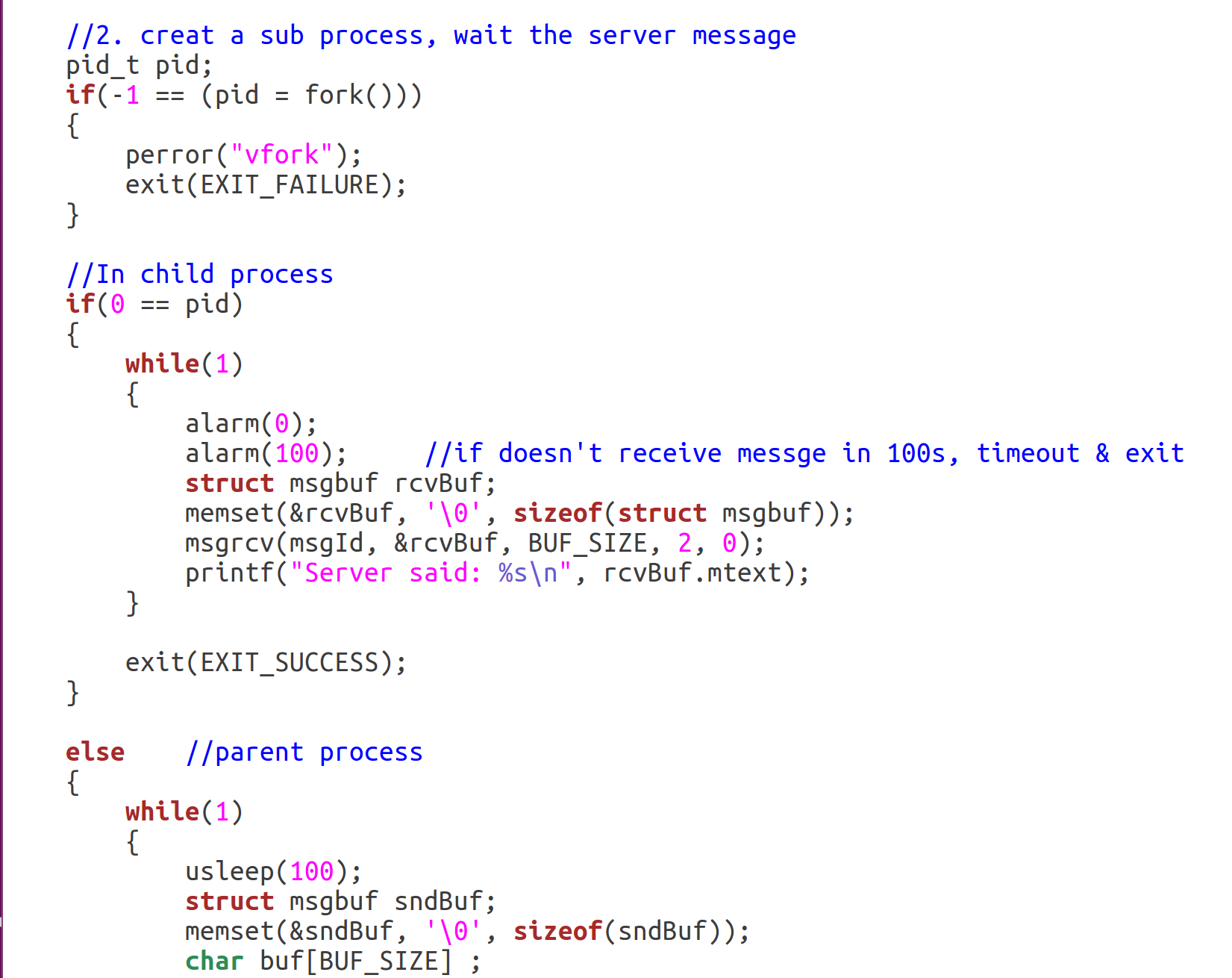
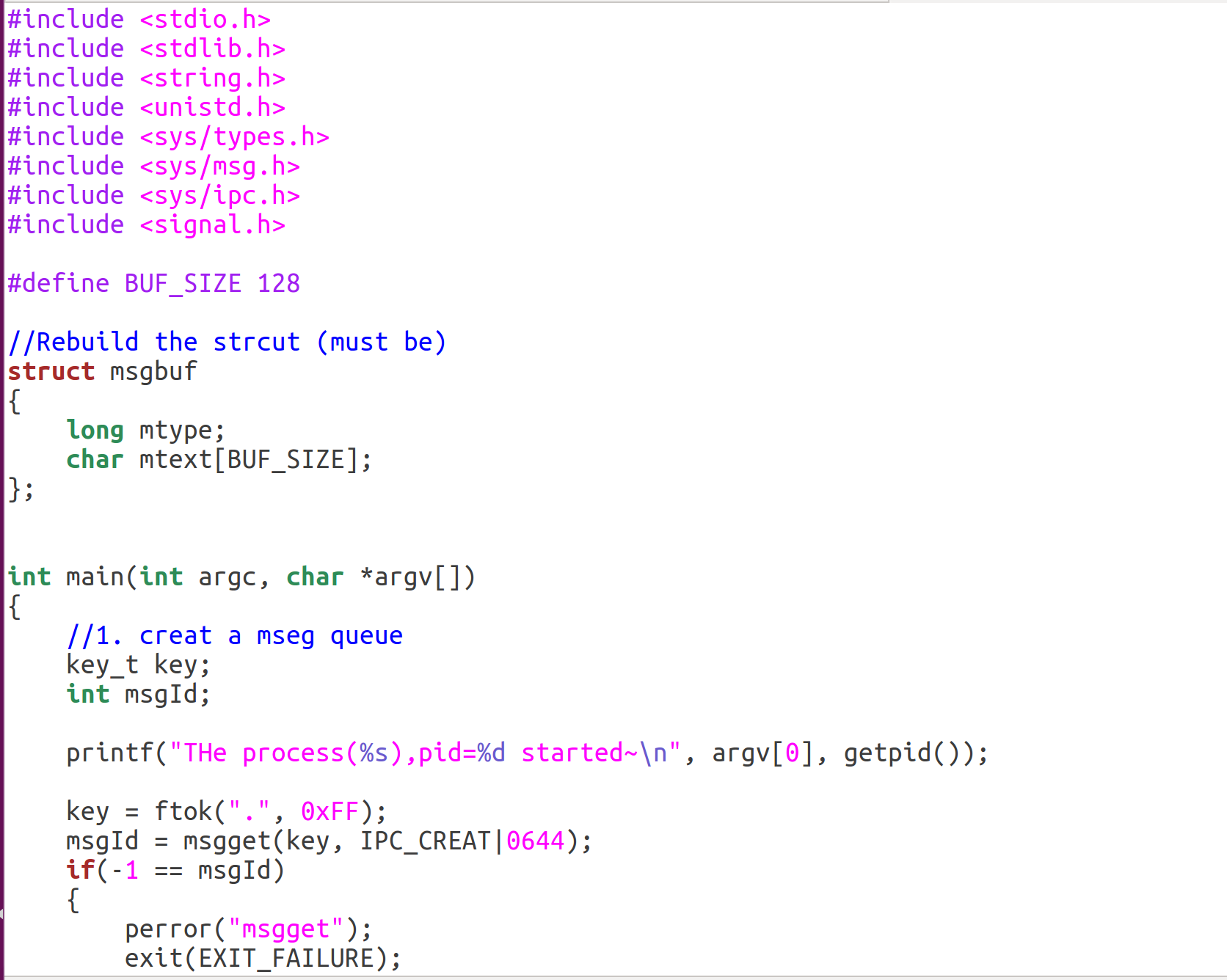


**c）消息通信系统调用是否已经实现了同步机制？通过实验验证，发送者和接收者如何同步的。比如，在什么情况下，发送者会阻塞，什么情况下，接收者会阻塞？**

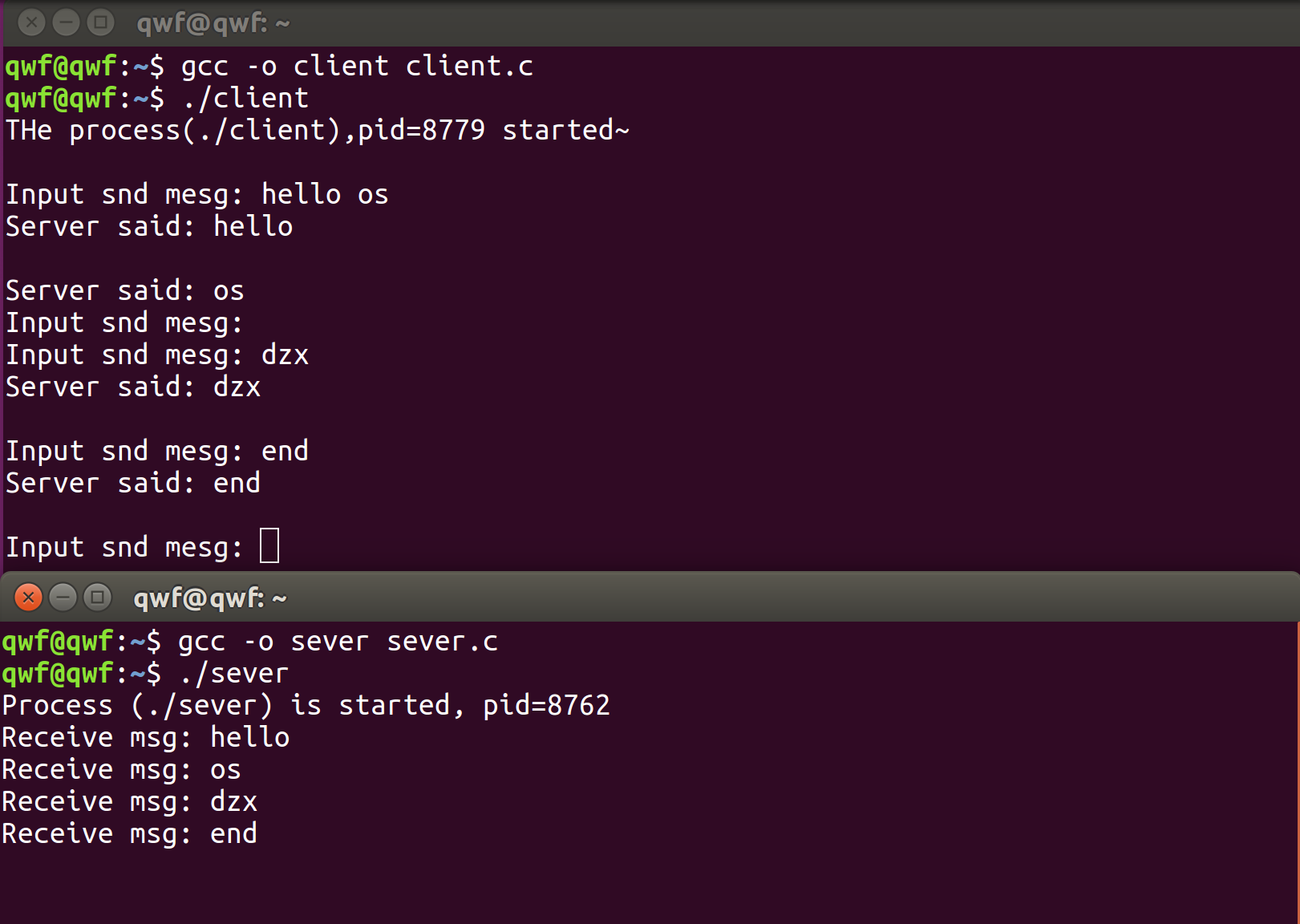
Sever.c:



Client.c：



运行结果如下：



客户端的子进程主要负责消息的接受，父进程主要负责消息的发送；

通过分析上面的代码可以知道，客户端和服务器端都是以阻塞的方式读取和写入消息。