

# 实验五：驱动程序问题

wangqr

## 0. 问题描述

管道是现代操作系统中重要的进程间通信（IPC）机制之一，Linux 和 Windows 操作系统都支持管道。

管道在本质上就是在进程之间以字节流方式传送信息的通信通道，每个管道具有两个端，一端用于输入，一端用于输出，如下图所示。在相互通信的两个进程中，一个进程将信息送入管道的输入端，另一个进程就可以从管道的输出端读取该信息。显然，管道独立于用户进程，所以只能在内核态下实现。

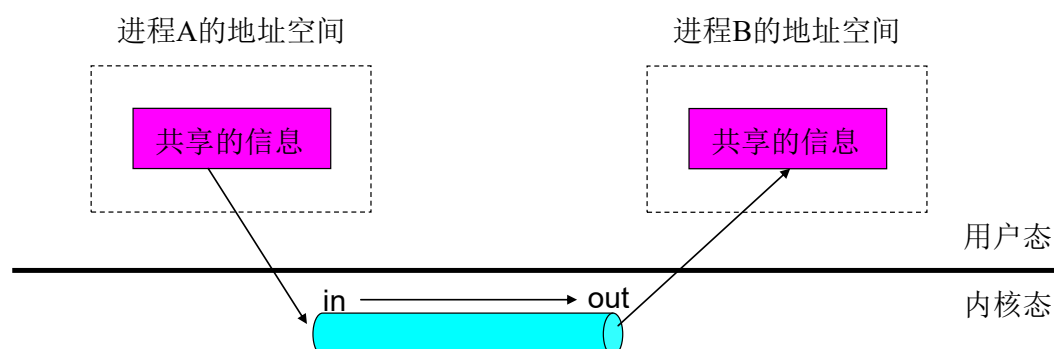


图 1 管道的工作原理

在本实验中，请通过编写设备驱动程序 `mypipe` 实现自己的管道，并通过该管道实现进程间通信。

你需要编写一个设备驱动程序 `mypipe` 实现管道，该驱动程序创建两个设备实例，一个针对管道的输入端，另一个针对管道的输出端。另外，你还需要编写两个测试程序，一个程序向管道的输入端写入数据，另一个程序从管道的输出端读出数据，从而实现两个进程间通过你自己实现的管道进行数据通信。

## 1. 实现方式

首先需要构建一个虚拟设备，因为管道是一个可读可写的处理字符流的设备，我们将其实现成一个字符设备（`char device`, `chrdev`），占据设备节点 `/dev/mypipe`。由于并不需要高级的字符设备功能，为了简化编程，将其实现为字符设备中特殊的一类：杂项设备（`misc device`），所有杂项设备共用主设备号 10，我们对次设备号也没有特殊需求，因此交由系统分配。

注册了设备后，需要实现对该设备操作的回调函数。这里允许设备被多个进程同时打开，因此对打开和关闭操作不作特殊实现。对于一些不适用的操作（如 `seek` 等）也不作实现。只实现写入操作和读取操作。

管道的基本思路是在内核态维护一个缓冲区，当有数据进入时写入该缓冲区，当有读取请求时从缓冲区读出。缓冲区的大小决定了可以暂存数据的量。

注意到 `*nix` 的读取和写入操作附带 `count` 参数，且需要返回一个结果，具体逻辑为：

- \* 当可读（写）的数据量大于或等于传入的 `count` 时，读（写）`count` 字节，并返回 `count` 本身；

- \* 当可读（写）的数据量小于传入的 `count` 但非零时，读（写）可用的部分，并返回成

功读（写）的字节数；

\* 当可读（写）的数据量为 0 但稍后可用时，将读（写）操作阻塞至可用，并按前两条处理。

为了符合上述要求，我们使用缓冲区满、缓冲区空两个互斥锁实现必要的阻塞操作（也可用信号量表示字节数，但开销过大）。当写入数据后，释放缓冲区空互斥锁，并判断是否需要将缓冲区满加锁。读取数据恰好相反。

## 2. 代码实现

缓冲区开设在内核态，大小可以任意指定，这里取为 4096 字节。声明了 4 个互斥锁，分别是读、写互斥锁（防止多个进程同时读或同时写），和缓冲区空、满互斥锁（用于在相应的情形下阻塞读、写操作）。缓冲区用数组实现，数组循环使用（即到达数组尾部后从头开始继续使用），使用两个指针指向数据部分的开头和空闲部分的开头。

```
#define KBUF_SIZE 4096

static char kbuf[KBUF_SIZE];
struct mutex m_read, m_write, m_empty, m_full;
static char *pstart, *pend;
```

读取操作时，先对读互斥锁加锁，再对缓冲区空加锁，均成功后即可开始读取。为了编程方便，当缓冲区数据部分从末尾环回到开头时，我们只读取末尾的部分，开头部分数据下次 read 时再读取。读取完成后，更新指向数据部分的开头的指针，必要时对缓冲区空加锁，并解锁缓冲区满（读取成功时缓冲区一定不是满的），最后解锁读互斥锁使其他读系统调用可以进行。

```
static ssize_t
mypipe_read (struct file *filp, char __user *buf, size_t count, loff_t *f_pos) {
    char *spend;
    int errno;
    if (count == 0)
        return 0;
    if (mutex_lock_interruptible(&m_read))
        return -ERESTARTSYS;
    if ((errno = mutex_lock_interruptible(&m_empty))) {
        mutex_unlock(&m_read);
        return -ERESTARTSYS;
    }
    spend = pend;
    if (spend <= pstart) { // used kbuf wrapped, read from unwrapped part only.
        if (count > KBUF_SIZE - (pstart - kbuf))
            count = KBUF_SIZE - (pstart - kbuf);
    }
    else if (count > spend - pstart)
        count = spend - pstart;
    if (copy_to_user(buf, pstart, count)) {
```

```

        mutex_unlock(&m_empty);
        mutex_unlock(&m_read);
        return -EINVAL;
    }
    pstart += count;
    if (pstart - kbuf == KBUF_SIZE)
        pstart = kbuf;
    if (pstart != spend)
        mutex_unlock(&m_empty);
    mutex_unlock(&m_full);
    mutex_unlock(&m_read);
    return count;
}

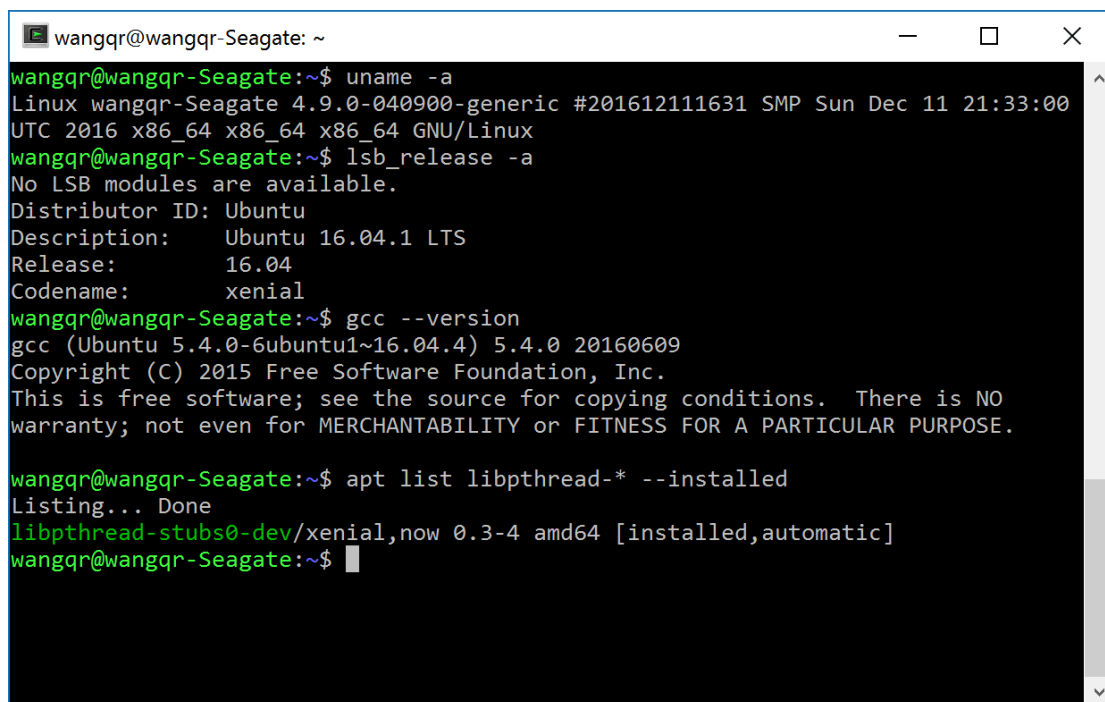
```

写操作与读操作结构完全相同，只需将读、写互斥锁互换，缓冲区空、满互斥锁互斥锁互换，两个缓冲区指针互换，内核态/用户态内存复制的方向调换即可。

完整的代码在 `mypipe.c` 中，同一文件夹下 `Makefile` 用于正确编译内核模块，编译完成后将产生 `mypipe.ko` 模块，可通过 `insmod` 加载。`demo.sh` 提供了从编译至模块加载，至 `mypipe` 的使用的完整演示。演示内容是将一个 256M 的随机文件经由 `mypipe` 发送，并比较发送和接收到的数据是否相同。

### 3. 运行结果

在如下的系统环境中对所编写的程序进行了测试：



```

wangqr@wangqr-Seagate: ~
wangqr@wangqr-Seagate:~$ uname -a
Linux wangqr-Seagate 4.9.0-040900-generic #201612111631 SMP Sun Dec 11 21:33:00
UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
wangqr@wangqr-Seagate:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:        16.04
Codename:       xenial
wangqr@wangqr-Seagate:~$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

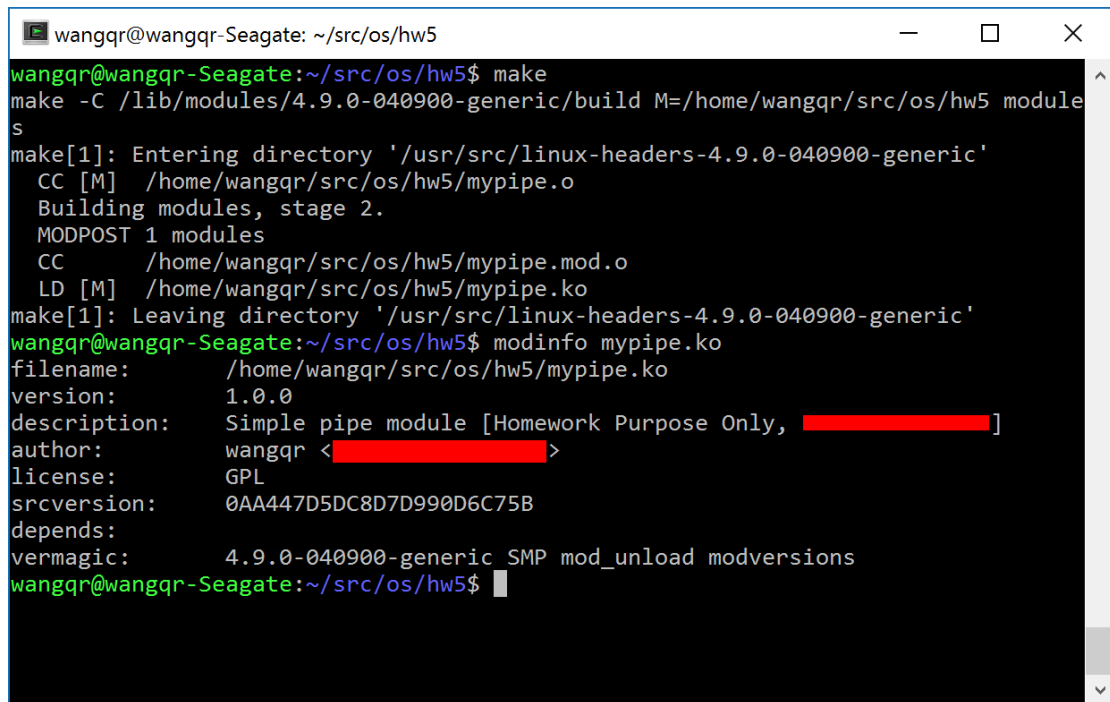
wangqr@wangqr-Seagate:~$ apt list libpthread-* --installed
Listing... Done
libpthread-stubs0-dev/xenial,now 0.3-4 amd64 [installed,automatic]
wangqr@wangqr-Seagate:~$

```

图 2 测试机器系统环境

使用 `make` 命令编译模块。编译完成后，我们可以通过 `modinfo` 指令查看编译出的内核

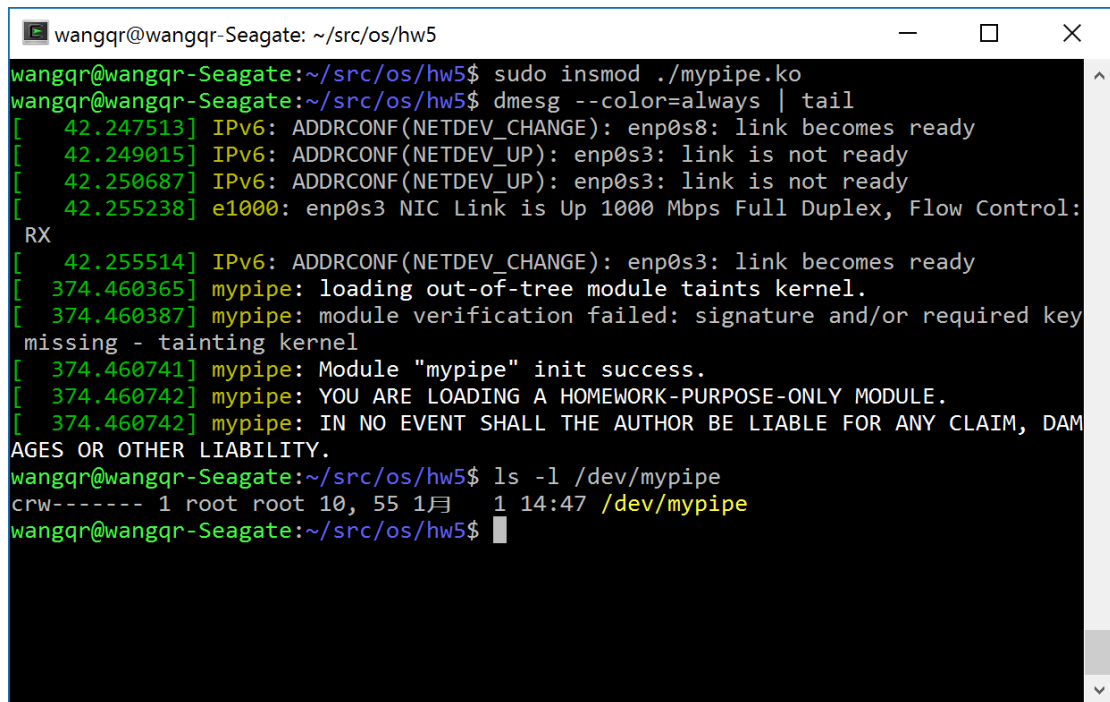
模块的基本信息。



```
wangqr@wangqr-Seagate: ~/src/os/hw5
wangqr@wangqr-Seagate:~/src/os/hw5$ make
make -C /lib/modules/4.9.0-040900-generic/build M=/home/wangqr/src/os/hw5 module
s
make[1]: Entering directory '/usr/src/linux-headers-4.9.0-040900-generic'
CC [M] /home/wangqr/src/os/hw5/mypipe.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/wangqr/src/os/hw5/mypipe.mod.o
LD [M] /home/wangqr/src/os/hw5/mypipe.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.0-040900-generic'
wangqr@wangqr-Seagate:~/src/os/hw5$ modinfo mypipe.ko
filename: /home/wangqr/src/os/hw5/mypipe.ko
version: 1.0.0
description: Simple pipe module [Homework Purpose Only, ]
author: wangqr < >
license: GPL
srcversion: 0AA447D5DC8D7D990D6C75B
depends:
vermagic: 4.9.0-040900-generic SMP mod_unload modversions
wangqr@wangqr-Seagate:~/src/os/hw5$
```

图 3 编译过程的输出

使用 insmod 加载内核模块。加载后通过查看系统日志（dmesg）可以看到，在代码中通过 kprintf 输出的调试信息显示模块已经正确被加载（之前的几行信息是说，该内核模块没有通过签名验证，由于该模块仅供作业用，因此没有对其签名）。加载之后，通过 ls 也可以看到，/dev/mypipe 已经存在，且它是一个字符设备。

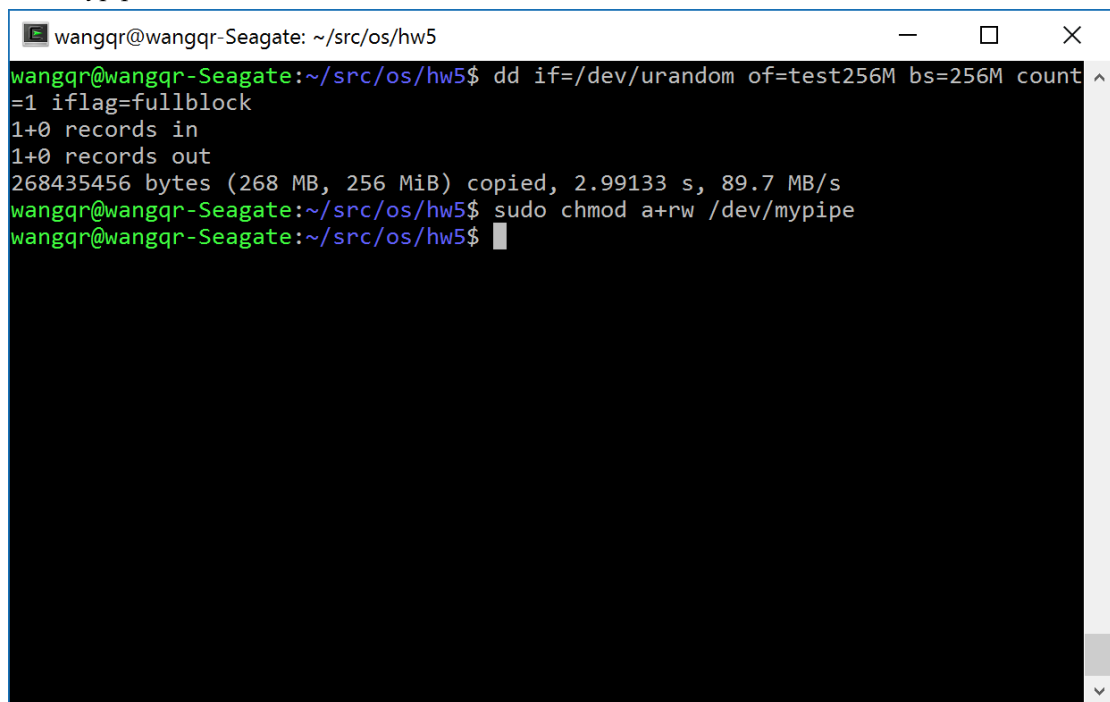


```
wangqr@wangqr-Seagate: ~/src/os/hw5
wangqr@wangqr-Seagate:~/src/os/hw5$ sudo insmod ./mypipe.ko
wangqr@wangqr-Seagate:~/src/os/hw5$ dmesg --color=always | tail
[ 42.247513] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s8: link becomes ready
[ 42.249015] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[ 42.250687] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[ 42.255238] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
RX
[ 42.255514] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[ 374.460365] mypipe: loading out-of-tree module taints kernel.
[ 374.460387] mypipe: module verification failed: signature and/or required key
missing - tainting kernel
[ 374.460741] mypipe: Module "mypipe" init success.
[ 374.460742] mypipe: YOU ARE LOADING A HOMEWORK-PURPOSE-ONLY MODULE.
[ 374.460742] mypipe: IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY CLAIM, DAM
AGES OR OTHER LIABILITY.
wangqr@wangqr-Seagate:~/src/os/hw5$ ls -l /dev/mypipe
crw----- 1 root root 10, 55 1月 1 14:47 /dev/mypipe
wangqr@wangqr-Seagate:~/src/os/hw5$
```

图 4 加载模块

我们对 mypipe 设备进行进一步测试前，先创建一个大小为 256M 的随机文件，并将

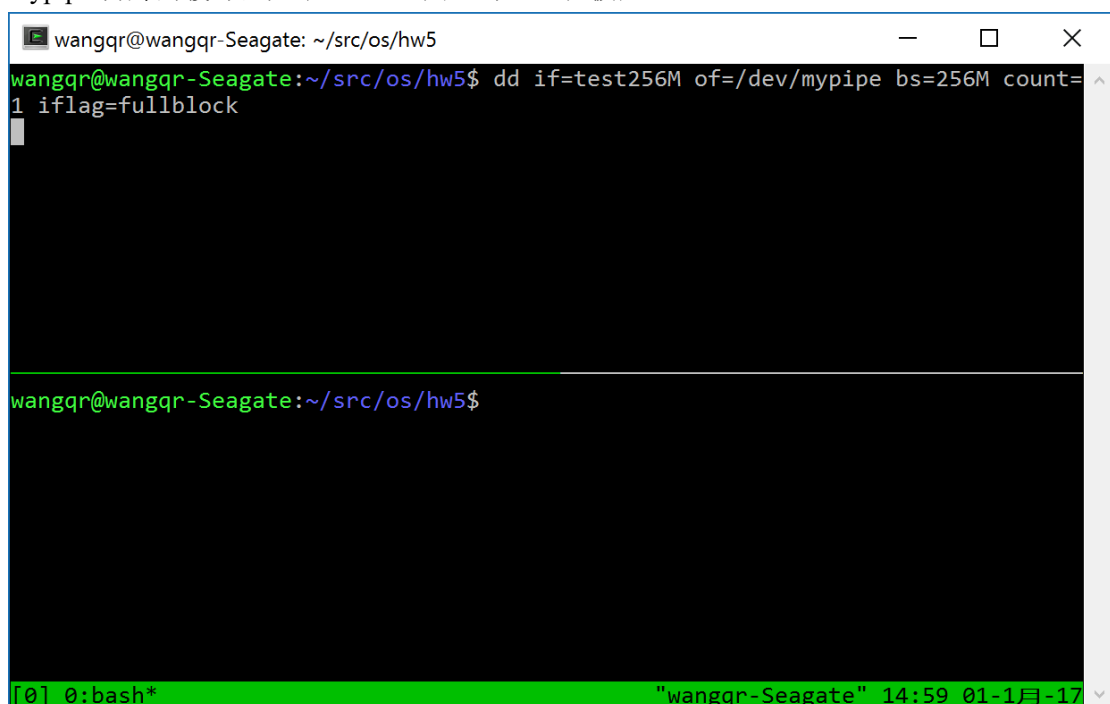
/dev/mypipe 赋予所有用户以读写权限。



```
wangqr@wangqr-Seagate: ~/src/os/hw5
wangqr@wangqr-Seagate:~/src/os/hw5$ dd if=/dev/urandom of=test256M bs=256M count=1 iflag=fullblock
1+0 records in
1+0 records out
268435456 bytes (268 MB, 256 MiB) copied, 2.99133 s, 89.7 MB/s
wangqr@wangqr-Seagate:~/src/os/hw5$ sudo chmod a+r /dev/mypipe
wangqr@wangqr-Seagate:~/src/os/hw5$
```

图 5 测试准备工作

我们使用 `dd` 命令将刚才产生的测试文件写入 `mypipe`，由于测试文件（256M）远大于 `mypipe` 自身的缓冲区大小（4k），因此写入过程被阻塞。



```
wangqr@wangqr-Seagate: ~/src/os/hw5
wangqr@wangqr-Seagate:~/src/os/hw5$ dd if=test256M of=/dev/mypipe bs=256M count=1 iflag=fullblock
^
wangqr@wangqr-Seagate:~/src/os/hw5$
```

[0] 0: bash\* "wangqr-Seagate" 14:59 01-1月-17

图 6 管道的写入阻塞

我们用另一个 `dd` 命令从 `mypipe` 中读取数据，这时可以发现，写入和读取都顺利完成了。通过比对发送和接收的数据的 SHA256 可以看出，传输过程没有差错。

```
wangqr@wangqr-Seagate: ~/src/os/hw5
wangqr@wangqr-Seagate:~/src/os/hw5$ dd if=test256M of=/dev/mypipe bs=256M count=1 iflag=fullblock
1+0 records in
1+0 records out
268435456 bytes (268 MB, 256 MiB) copied, 22.4013 s, 12.0 MB/s
wangqr@wangqr-Seagate:~/src/os/hw5$ sha256sum test256M
59e0b1df88a88fba3a715fe6d2681e4b6b064271f9384b101a8faeaa800726c5 test256M
wangqr@wangqr-Seagate:~/src/os/hw5$

wangqr@wangqr-Seagate:~/src/os/hw5$ dd if=/dev/mypipe of=recv256M bs=256M count=1 iflag=fullblock
1+0 records in
1+0 records out
268435456 bytes (268 MB, 256 MiB) copied, 2.36064 s, 114 MB/s
wangqr@wangqr-Seagate:~/src/os/hw5$ sha256sum recv256M
59e0b1df88a88fba3a715fe6d2681e4b6b064271f9384b101a8faeaa800726c5 recv256M
wangqr@wangqr-Seagate:~/src/os/hw5$
```

图 7 接收数据比较

我们也可以用多个进程同时读取 mypipe，下图中左侧两个进程同时读取，右侧一个进程写入。可以看到，每份数据被且只被一个进程读取到一次，与设计相符。（右侧第二行的 pe 是指令 cat >/dev/mypipe 的一部分；由于 stdin 的行缓冲机制，数据是按照行写入 /dev/mypipe 的，又由于数据量很小，没有使 mypipe 的缓冲区环回，所以读取必定可以读取到整行字符而不会在行中断开）

```
wangqr@wangqr-Seagate: ~
wangqr@wangqr-Seagate:~$ cat /dev/mypipe
yu
iuh
gf
ds
c
vbn
hg
fd

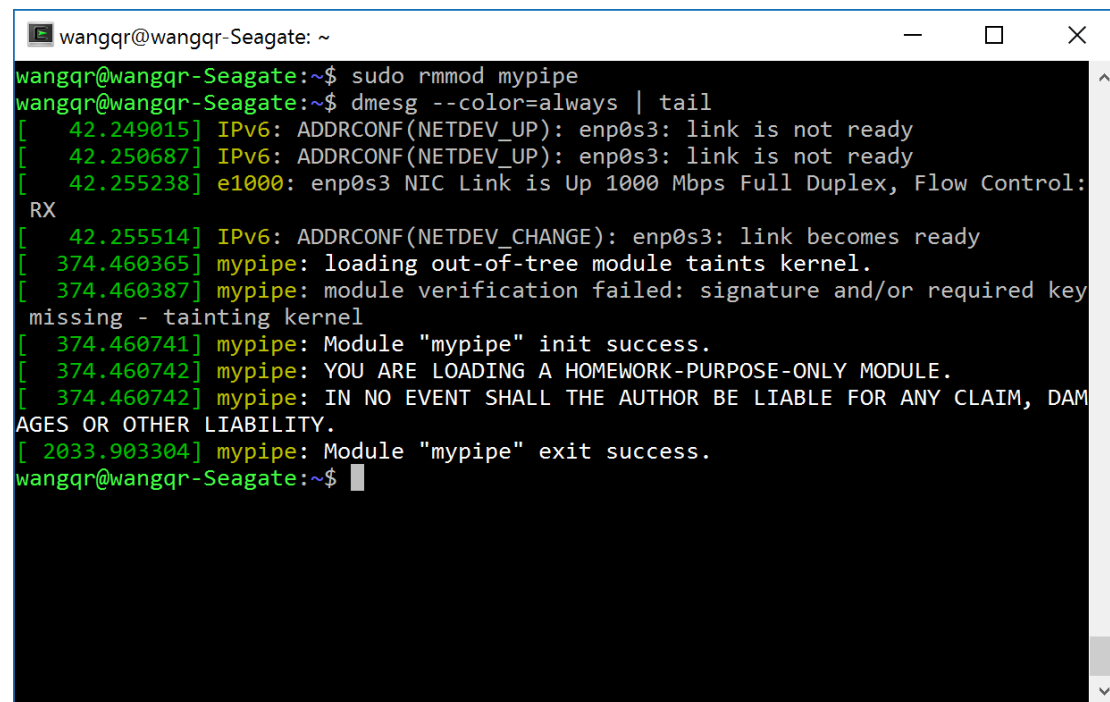
wangqr@wangqr-Seagate:~$ cat /dev/mypipe
q
wert
i
o

wangqr@wangqr-Seagate:~$ cat >/dev/mypipe
pe
q
wert
yu
i
o
iuh
gf
ds
c
vbn
hg
fd
```

图 8 多进程同时读取的测试

最后，我们可以通过 rmmmod 卸载 mypipe 模块。卸载后同样可以在系统日志（dmesg）

中看到我们在卸载程序中输出的调试信息，以确认卸载成功。

A terminal window titled 'wangqr@wangqr-Seagate: ~' with standard window controls. The terminal shows the execution of 'sudo rmmod mypipe' followed by 'dmesg --color=always | tail'. The output displays various kernel messages, including network status updates and the successful unloading of the 'mypipe' module. The messages are color-coded: green for timestamps and 'IPv6', yellow for 'e1000', and red for the module's warning messages.

```
wangqr@wangqr-Seagate: ~  
wangqr@wangqr-Seagate:~$ sudo rmmod mypipe  
wangqr@wangqr-Seagate:~$ dmesg --color=always | tail  
[ 42.249015] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready  
[ 42.250687] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready  
[ 42.255238] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX  
[ 42.255514] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready  
[ 374.460365] mypipe: loading out-of-tree module taints kernel.  
[ 374.460387] mypipe: module verification failed: signature and/or required key missing - tainting kernel  
[ 374.460741] mypipe: Module "mypipe" init success.  
[ 374.460742] mypipe: YOU ARE LOADING A HOMEWORK-PURPOSE-ONLY MODULE.  
[ 374.460742] mypipe: IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY.  
[ 2033.903304] mypipe: Module "mypipe" exit success.  
wangqr@wangqr-Seagate:~$
```

图9 卸载模块