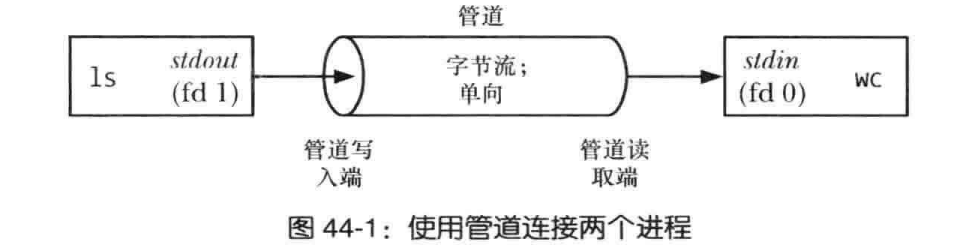
第四十四章

1. 管道和FIFO：
2. 管道概述：

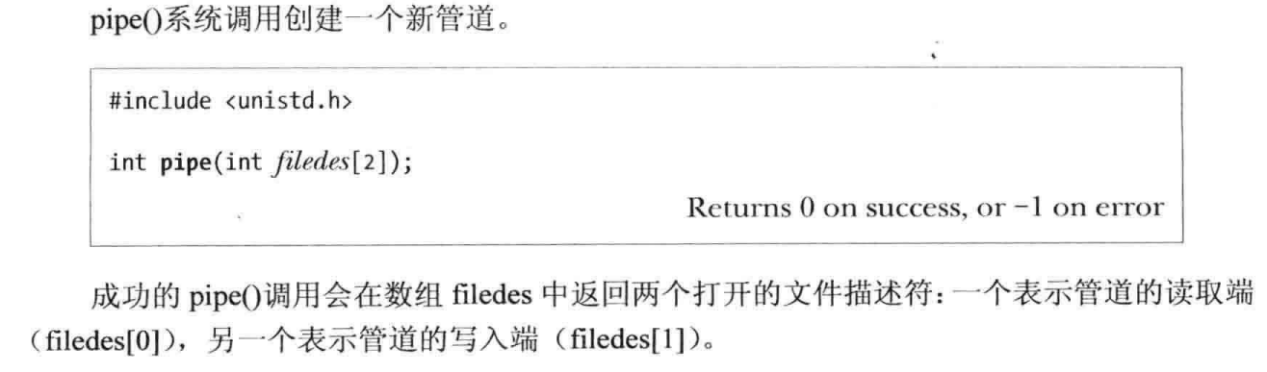
$ ls | wc -l

下图展示这两个命令如何使用管道：

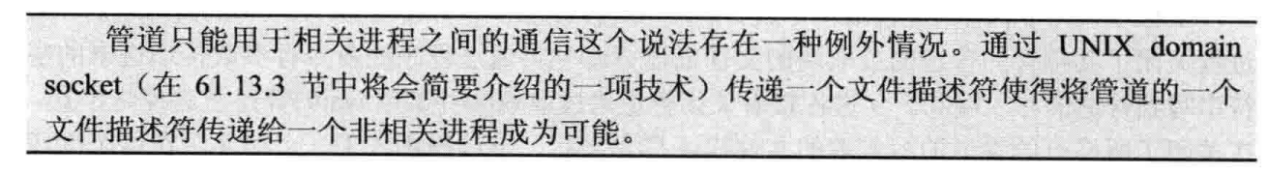


管道的重要特征：（管道是在内核内存中维护的缓冲器）

1. 一个管道是一个字节流。
2. 可以从管道中读取数据。
3. 管道是单向的。
4. 可以确保写入不超过PIPE\_BUF字节的操作是原子的。
5. 管道的容量是有限的。
6. 创建和使用管道：



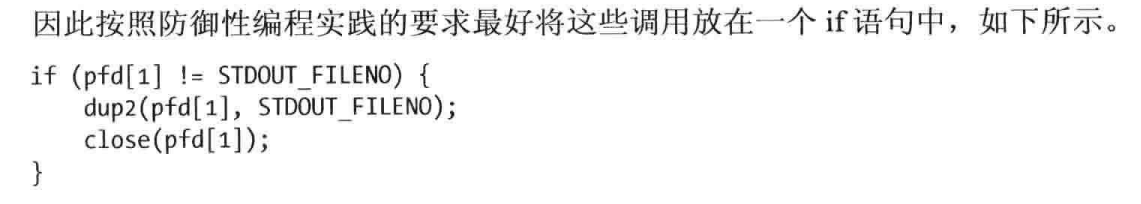
管道允许相关进程的通信，这里的相关是指拥有共同的祖先创建管道即可。但是还有另外一种情况：



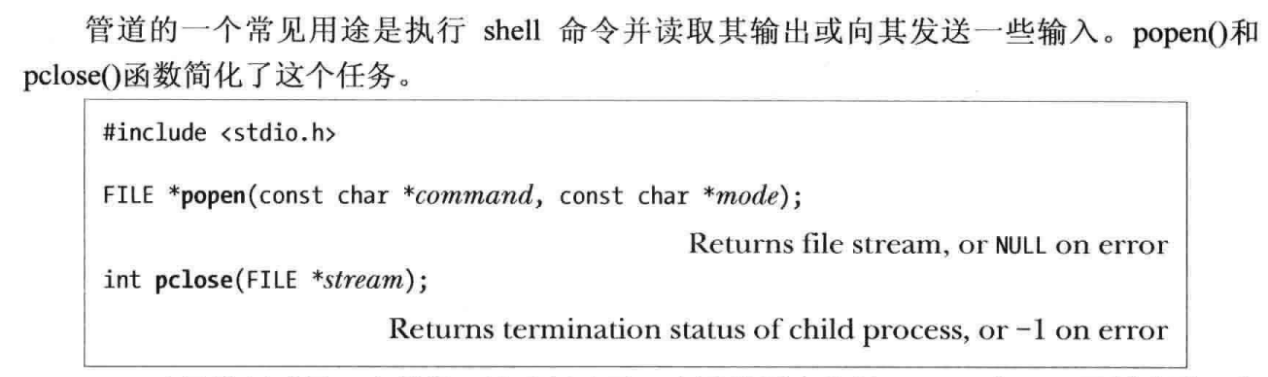
作为好的编程习惯，要关闭不使用的管道文件描述符。

1. 可以将管道作为一种进程同步的方法：
2. 使用管道连接过滤器：

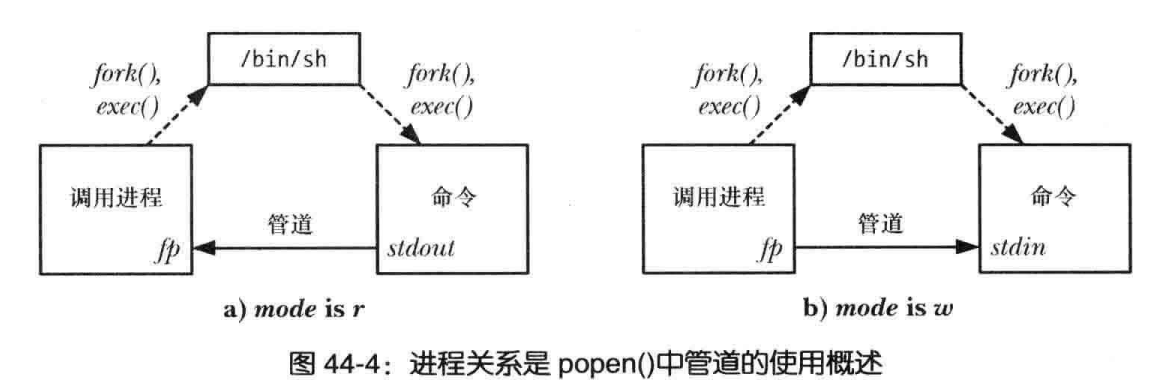
关键代码：



1. 通过管道与shell命令行进行通信：



使用popen的进程关系：



不过我运行的时候使用pstree查看进程没有发现有/bin/sh这个进程。

关闭管道应该使用pclose而不是fclose。因为要等待子进程结束。

Pclose在成功的时候返回终止状态，与26.1.3里描述的一样。

这个函数跟system非常相似，不过有显著差异：popen是调用进程和命令是并行运行，具体包括两个方面：

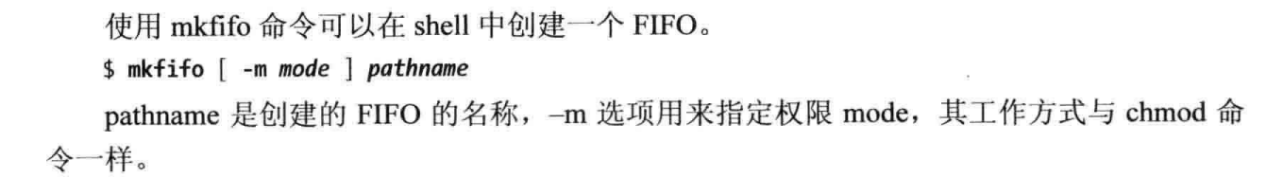
1. 不能忽略SIGINT和SIGQUIT信号。
2. 在popen和pclose直接调用wait导致后者无法获得子进程状态。
3. 管道和缓冲：

由于popen调用返回的文件流指针没有引用终端，所以stdio库采用行缓冲。如果调用进程是写端，则可以使用setbuf或者fflush。相反，如果调用进程是读端，则措施比较有限。可以使用64章介绍的伪终端。

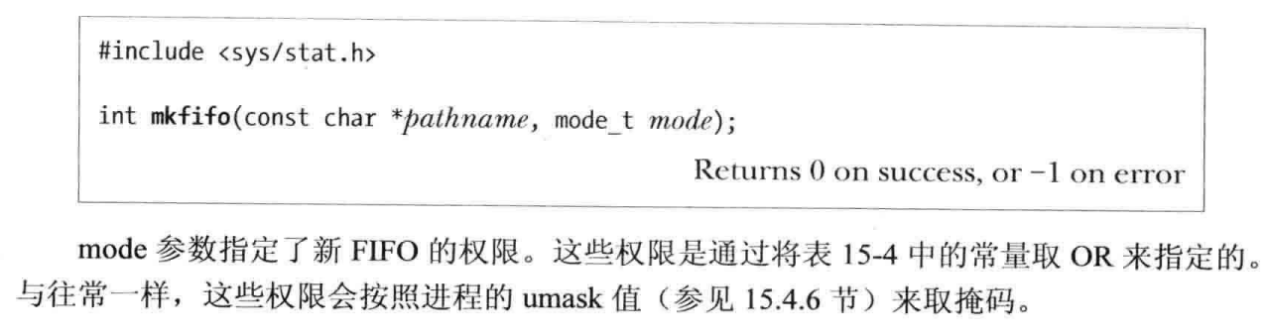
1. FIFO：

又称命名管道，语义上和管道类似，但是在文件系统中拥有名称。

在shell中创建一个FIFO：

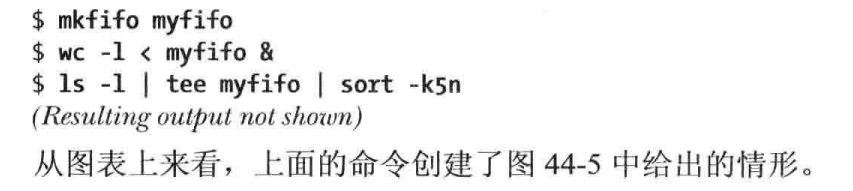


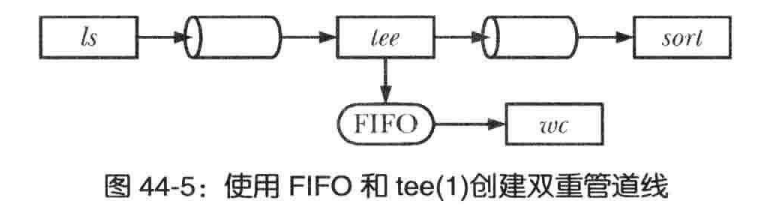
也可以使用函数：



打开一个FIFO文件具备不寻常的语义。一端以读打开会阻塞直到另一端以写打开。相反同理。指定O\_RDWR可以绕过阻塞行为，但是从可移植性来说尽量避免。可以使用标准化标记O\_NONBLOCK解决这一问题。

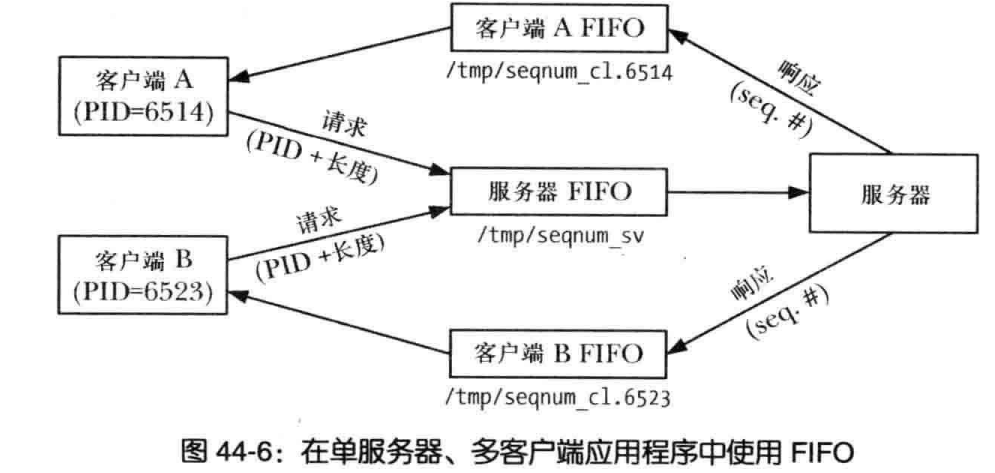
使用FIFO和tee创建双重管道线：





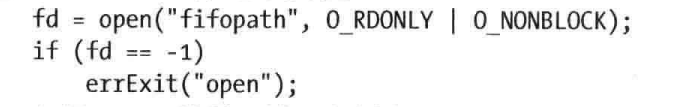
1. 使用管道实现一个客户端和服务端程序：

基本架构是这样的：



1. 非阻塞IO：

使用方式：



注意，在读取下才能成功，用于写入的时候open调用失败并且errno设置为ENXIO。



使用非阻塞的目的：

1. 允许一个进程打开两端。
2. 防止死锁。

对于打开了的文件描述符，可以使用下图所示进行修改。



1. 管道和FIFO里的read含义：



