## 实验五；管道通信

返回[目录](#_目录)

### 实验目的：

1. 掌握管道的通信原理与过程
2. 可以使用管道实现进程间通信

### 预备知识：

1. 概念

管道有两种类型：无名管道和有名管道。无名管道没有名字，也从来不在文件系统中出现，只是和内存中的一个索引节点相关联的两个文件描述符，该索引节点指向指向内存中的一个物理块。写进程向管道写入数据时，字节被复制到共享数据页面中，读进程从管道读出数据时，字节从共享页面中读出。无名管道是半双工的，数据只能在一个方向传送，而且只能在**相关的、有共同祖先的的进程**之间使用。

有（命）名管道（FIFO）可以为两个不相关的进程提供通信。它不是临时对象，是文件系统中的一个实体，可以用mknod和mkfifo创建。在写进程使用之前，必须让读进程先打开管道，任何读进程从中读取之前必须有写进程向其写入数据。FIFO有一个路径与之关联，故无亲缘关系的进程可以访问同一个FIFO。

1. 无名管道工作原理

无名管道由单个进程创建，但很少在单个进程内使用。其典型用途是在一个父进程和子进程之间通信。首先由一个进程创建一个管道后调用fork派生一个自身的拷贝，如图1：

进程

数据流

内核

子进程

父进程

fork()

fd[0]

fd[1]

管道

fd[0]

fd[1]

图1：单个进程内的管道

然后，父进程关闭该管道的读出端，子进程关闭同一管道的写入端，这样就在父子进程间提供了一个单项数据流，如图2。

进程

内核

数据流

子进程

父进程

fd[0]

fd[1]

管道

fd[0]

fd[1]

图2：两个进程间的管道

当需要一个双向数据流时，必须建立两个管道，每个方向一个，实际步骤如下：

* 1. 创建管道1（fd1[0]和fd1[1]）和管道2（fd2[0]和fd2[1]）
  2. fork()
  3. 父进程关闭管道1的读出端fd1[0]
  4. 父进程关闭管道2的写入端fd2[1]
  5. 子进程关闭管道1的写入端fd1[1]
  6. 子进程关闭管道2的读出端fd2[0]

管道布局如图3：

图3：提供一个双向数据流的两个管道

进程

数据流

内核

子进程

父进程

fd1[1]

fd2[0]

管道1

fd2[1]

fd1[0]

数据流

管道2

管道读写分别使用read和write系统调用，其中读取字节数不应大于PIPE\_BUF(<limits.h>中定义，主要是为了保证原子操作)，因此通常定义缓冲区大小为PIPE\_BUF。管道使用完后要关闭(close)。

1. 基本命令

3.1 $command1 | command2

将command1的标准输出作为command2的标准输入。

* 1. mknod 创建块、字符或管道文件
  2. mkfifo 创建一个命名管道(FIFO)

1. 系统调用

4.1 pipe（建立管道）

表头文件 #include<unistd.h>

定义函数 int pipe(int filedes[2]);

函数说明 pipe()会建立管道，并将文件描述词由参数filedes数组返回。filedes[0]为管道里的读取端，filedes[1]则为管道的写入端。

返回值 若成功则返回零，否则返回-1，错误原因存于errno中。

错误代码 EMFILE 进程已用完文件描述词最大量。

ENFILE 系统已无文件描述词可用。

EFAULT 参数filedes数组地址不合法。

4.2 mkfifo（建立有名管道）

表头文件 #include<sys/types.h>

#include<sys/stat.h>

定义函数 int mkfifo(const char \* pathname,mode\_t mode);

函数说明 mkfifo()会依参数pathname建立特殊的FIFO文件，该文件必须不存在，而参数mode为该文件的权限（mode&~umask），因此umask值也会影响到FIFO文件的权限。mkfifo()建立的FIFO文件其他进程都可以用读写一般文件的方式存取。当使用open()来打开FIFO文件时，O\_NONBLOCK旗标会有影响

1、当使用O\_NONBLOCK 旗标时，打开FIFO 文件来读取的操作会立刻返回，但是若还没有其他进程打开FIFO 文件来读取，则写入的操作会返回ENXIO 错误代码。

2、没有使用O\_NONBLOCK 旗标时，打开FIFO 来读取的操作会等到其他进程打开FIFO文件来写入才正常返回。同样地，打开FIFO文件来写入的操作会等到其他进程打开FIFO 文件来读取后才正常返回。

返回值 若成功则返回0，否则返回-1，错误原因存于errno中。

错误代码 EACCESS 参数pathname所指定的目录路径无可执行的权限

EEXIST 参数pathname所指定的文件已存在。

ENAMETOOLONG 参数pathname的路径名称太长。

ENOENT 参数pathname包含的目录不存在

ENOSPC 文件系统的剩余空间不足

ENOTDIR 参数pathname路径中的目录存在但却非真正的目录。

EROFS 参数pathname指定的文件存在于只读文件系统内。

4.3 open（打开文件）

表头文件 #include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

定义函数 int open( const char \* pathname, int flags);

int open( const char \* pathname,int flags, mode\_t mode);

函数说明 参数pathname 指向欲打开的文件路径字符串。下列是参数flags 所能使用的旗标:

O\_RDONLY 以只读方式打开文件

O\_WRONLY 以只写方式打开文件

O\_RDWR 以可读写方式打开文件。上述三种旗标是互斥的，也就是不可同时使用，但可与下列的旗标利用OR(|)运算符组合。

O\_CREAT 若欲打开的文件不存在则自动建立该文件。

O\_EXCL 如果O\_CREAT 也被设置，此指令会去检查文件是否存在。文件若不存在则建立该文件，否则将导致打开文件错误。此外，若O\_CREAT与O\_EXCL同时设置，并且欲打开的文件为符号连接，则会打开文件失败。

返回值 若所有欲核查的权限都通过了检查则返回0 值，表示成功，只要有一个权限被禁止则返回-1。

错误代码 EEXIST 参数pathname 所指的文件已存在，却使用了O\_CREAT和O\_EXCL旗标。

EACCESS 参数pathname所指的文件不符合所要求测试的权限。

EROFS 欲测试写入权限的文件存在于只读文件系统内。

EFAULT 参数pathname指针超出可存取内存空间。

EINVAL 参数mode 不正确。

ENAMETOOLONG 参数pathname太长。

ENOTDIR 参数pathname不是目录。

ENOMEM 核心内存不足。

ELOOP 参数pathname有过多符号连接问题。

EIO I/O 存取错误。

4.4 close（关闭文件）

表头文件 #include<unistd.h>

定义函数 int close(int fd);

函数说明 当使用完文件后若已不再需要则可使用close()关闭该文件，二close()会让数据写回磁盘，并释放该文件所占用的资源。参数fd为先前由open()或creat()所返回的文件描述词。

返回值 若文件顺利关闭则返回0，发生错误时返回-1。

错误代码 EBADF 参数fd 非有效的文件描述词或该文件已关闭。

4.5 read（由已打开的文件读取数据）

表头文件 #include<unistd.h>

定义函数 ssize\_t read(int fd,void \* buf ,size\_t count);

函数说明 read()会把参数fd 所指的文件传送count个字节到buf指针所指的内存中。若参数count为0，则read()不会有作用并返回0。返回值为实际读取到的字节数，如果返回0，表示已到达文件尾或是无可读取的数据，此外文件读写位置会随读取到的字节移动。

错误代码 EINTR 此调用被信号所中断。

EAGAIN 当使用不可阻断I/O 时（O\_NONBLOCK），若无数据可读取则返回此值。

EBADF 参数fd 非有效的文件描述词，或该文件已关闭。

4.6 write（将数据写入已打开的文件内）

表头文件 #include<unistd.h>

定义函数 ssize\_t write (int fd,const void \* buf,size\_t count);

函数说明 write()会把参数buf所指的内存写入count个字节到参数fd所指的文件内。当然，文件读写位置也会随之移动。

返回值 如果顺利write()会返回实际写入的字节数。当有错误发生时则返回-1，错误代码存入errno中。

错误代码 EINTR 此调用被信号所中断。

EAGAIN 当使用不可阻断I/O 时（O\_NONBLOCK），若无数据可读取则返回此值。

EADF 参数fd非有效的文件描述词，或该文件已关闭。

4.7 unlink（删除文件）

表头文件 #include<unistd.h>

定义函数 int unlink(const char \* pathname);

函数说明 unlink()会删除参数pathname指定的文件。如果该文件名为最后连接点，但有其他进程打开了此文件，则在所有关于此文件的文件描述词皆关闭后才会删除。如果参数pathname为一符号连接，则此连接会被删除。

返回值 成功则返回0，失败返回-1，错误原因存于errno

错误代码 EROFS 文件存在于只读文件系统内

EFAULT 参数pathname指针超出可存取内存空间

ENAMETOOLONG 参数pathname太长

ENOMEM 核心内存不足

ELOOP 参数pathname 有过多符号连接问题

EIO I/O 存取错误

1. 标准库函数

5.1 popen（建立管道I/O）

表头文件 #include<stdio.h>

定义函数 FILE \* popen( const char \* command,const char \* type);

函数说明 popen()会调用fork()产生子进程，然后从子进程中调用/bin/sh -c来执行参数command的指令。参数type可使用“r”代表读取，“w”代表写入。依照此type值，popen()会建立管道连到子进程的标准输出设备或标准输入设备，然后返回一个文件指针。随后进程便可利用此文件指针来读取子进程的输出设备或是写入到子进程的标准输入设备中。此外，所有使用文件指针(FILE\*)操作的函数也都可以使用，除了fclose()以外。

返回值 若成功则返回文件指针，否则返回NULL，错误原因存于errno中。

错误代码 EINVAL参数type不合法。

5.2 pclose（关闭管道I/O）

表头文件 #include<stdio.h>

定义函数 int pclose(FILE \* stream);

函数说明 pclose()用来关闭由popen所建立的管道及文件指针。参数stream为先前由popen()所返回的文件指针。

返回值 返回子进程的结束状态。如果有错误则返回-1，错误原因存于errno中。

错误代码 ECHILD pclose()无法取得子进程的结束状态。

### 实验内容:

1. 阅读以下程序：

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

main()

{

int filedes[2];

char buffer[80];

if(pipe(filedes)<0)

err\_quit(“pipe error”);

if(fork()>0){

char s[ ] = “hello!\n”;

close(filedes[0]);

write(filedes[1],s,sizeof(s));

close(filedes[1]);

}else{

close(filedes[1]);

read(filedes[0],buffer,80);

printf(“%s”,buffer);

close(filedes[0]);

}

}

编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

1. 阅读以下程序：

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

main()

{

char buffer[80];

int fd;

unlink(FIFO);

mkfifo(FIFO,0666); //FIFO是管道名，0666是权限

if(fork()>0){

char s[ ] = “hello!\n”;

fd = open (FIFO,O\_WRONLY);

write(fd,s,sizeof(s));

close(fd);

}else{

fd= open(FIFO,O\_RDONLY);

read(fd,buffer,80);

printf(“%s”,buffer);

close(fd);

}

}

编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

1. 阅读以下程序：

#include<stdio.h>

main()

{

FILE \* fp;

char buffer[80];

fp=popen(“cat /etc/passwd”,”r”);

fgets(buffer,sizeof(buffer),fp);

printf(“%s”,buffer);

pclose(fp);

}

编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

1. 编写一个程序，读取一个数据文件，对每一个数据进行某种运算，再在屏幕输出计算结果。要求以上工作用两个进程实现，父进程负责读文件和显示，子进程进行计算，进程间通信使用无名管道。（使用系统调用）
2. 编写两个程序，一个创建一个FIFO，并读管道，并显示在屏幕上，另一个每过一段时间向该管道写数据（进程PID）。运行多个写程序和一个读程序，观察运行结果。（使用系统调用）
3. 结合第5题，分析程序pipe5-program。

### 实验报告：

完成上述要求中的第1,2,3题并提交报告。