

# pipe 函数 （C语言）

转载

qzwuji...

于 2010-11-22

17:10:00 发布

17535

17

收藏

文章标签: 语言 c cmd linux shell 数据结构

## pipe 函数 （C语言）

pipe我们用中文叫做管道。  
以下讲解均是基于Linux为环境：

### 函数简介

所需头文件 #include<unistd.h>  
函数原型 int pipe(int fd[2])  
函数传入值 fd[2]:管道的两个文件描述符，之后就是可以直接操作者两个文件描述符  
返回值 成功 0 失败 -1

### 什么是管道

管道是Linux 支持的最初Unix IPC形式之一，具有以下特点：  
管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立起两个管道； 只能用于父子进程或者兄弟进程之间（具有亲缘关系的进程）； 单独构成一种独立的文件系统：管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系 统，并且只存在与内存中。数据的读出和写入：一个进程向管道中写的内容被管道另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾，并且每次都是从缓冲区的头部读出数据。

### 管道的创建

#include <unistd.h>  
int pipe(int fd[2])  
该函数创建的管道的两端处于一个进程中间，在实际应用中没有太大意义，因此，一个进程在由 pipe()创建管道后，一般再fork一个子进程，然后通过管道实现父子进程间的通信（因此也不难推出，只要两个进程中存在亲缘关系，这里的亲缘关系指的是具有共同的祖先，都可以采用管道方式来进行通信）。

### 管道的读写规则

管道两端可 分别用描述字fd[0]以及fd[1]来描述，需要注意的是，管道的两端是固定了任务的。即一端只能用于读，由描述字fd[0]表示，称其为管道读端；另 一端则只能用于写，由描述字fd[1]来表示，称其为管道写端。如果试图从管道写端读取数据，或者向管道读端写入数据都将导致错误发生。一般文件的I/O 函数都可以用于管道，如close、read、write 等等。  
从管道中读取数据：  
如果管道的写端不存在，则认为已经读到了数据的末尾，读函数返回的读出字节数为0； 当管道的写端存在时，如果请求的字节数目大于 PIPE\_BUF，则返回管道中现有的数据字节数，如果请求的字节数目不大于PIPE\_BUF，则返回管道中 现有数据字节数（此时，管道中数据量小于请求的数据量）； 或者返回请求的字节数（此时，管道中数据量不小于请求的数据量）。注：  
义，不同的内核版本可能会有

 qzwujiaying

关注

512字节，red hat 7.2中为4096）。

关于管道的读规则验证：

```

/*****
 * readtest.c *
 *****/

#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
main()
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[100];
    char w_buf[4];
    char* p_wbuf;
    int r_num;
    int cmd;
    memset(r_buf,0,sizeof(r_buf));
    memset(w_buf,0,sizeof(r_buf));
    p_wbuf=w_buf;
    if(pipe(pipe_fd)<0)
    {
        printf("pipe create error ");
        return -1;
    }
    if((pid=fork())==0)
    {
        printf(" ");
        close(pipe_fd[1]);
        sleep(3);//确保父进程关闭写端
        r_num=read(pipe_fd[0],r_buf,100);
        printf( "read num is %d the data read from the pipe is %d\n",r_num,atoi(r_buf));
        close(pipe_fd[0]);
        exit();
    }
    else if(pid>0)
    {
        close(pipe_fd[0]);//read
        strcpy(w_buf,"111");
        if(write(pipe_fd[1],w_buf,4)!=-1)
            printf("parent write over ");
        close(pipe_fd[1]);//write
        printf("parent close fd[1] over ");
        sleep(10);
    }
}

/*****
 * 程序输出结果:
 * parent write over
 * parent close fd[1] over
 * read num is 4 the data read from the pipe is 111
 * 附加结论:
 * 管道写端关闭后，写入的数据将一直存在，直到读出为止.
 *****/

```

向管道中写入数据：

向管道中写入数据时，linux将  
闲区域，写进程就会试图向



qzwujiaying

关注

10



17

区中的数据，那么写操作将一直阻塞。

注：只有在管道的读端存在时，向管道中写入数据才有意义。否则，向管道中写入数据的进程将收到 内核传来的SIGPIPE信号，应用程序可以处理该信号，也可以忽略（默认动作则是应用程序终止）。

对管道的写规则的验证1：写端对读端存在的依赖性

```
#include <unistd.h>
#include <sys/types.h>
main()
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[4];
    char* w_buf;
    int writenum;
    int cmd;
    memset(r_buf,0,sizeof(r_buf));
    if(pipe(pipe_fd)<0)
    {
        printf("pipe create error ");
        return -1;
    }
    if((pid=fork())==0)
    {
        close(pipe_fd[0]);
        close(pipe_fd[1]);
        sleep(10);
        exit();
    }
    else if(pid>0)
    {
        sleep(1); //等待子进程完成关闭读端的操作
        close(pipe_fd[0]); //write
        w_buf="111";
        if((writenum=write(pipe_fd[1],w_buf,4))!=-1)
            printf("write to pipe error ");
        else
            printf("the bytes write to pipe is %d ", writenum);
        close(pipe_fd[1]);
    }
}
```

则输出结果为： Broken pipe,原因就是该管道以及它的所有fork()产物的读端都已经被关闭。如果在父进程中保留读端，即在写完pipe后，再关闭父进程的读端，也会正常 写入pipe，读者可自己验证一下该结论。因此，在向管道写入数据时，至少应该存在某一个进程，其中管道读端没有被关闭，否则就会出现上述错误（管道断裂,进程收到了SIGPIPE信号，默认动作是进程终止）

对管道的写规则的验证2：linux不保证写管道的原子性验证

```
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
main(int argc,char**argv)
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[4096];
    char w_buf[4096*2];
    int writenum;
    int rnum;
```



qzwujiaying

关注

10



17

```

memset(r_buf,0,sizeof(r_buf));
if(pipe(pipe_fd)<0)
{
printf("pipe create error ");
return -1;
}
if((pid=fork())==0)
{
close(pipe_fd[1]);
while(1)
{
sleep(1);
rnum=read(pipe_fd[0],r_buf,1000);
printf("child: readnum is %d ",rnum);
}
close(pipe_fd[0]);
exit();
}
else if(pid>0)
{
close(pipe_fd[0]);//write
memset(r_buf,0,sizeof(r_buf));
if((writenum=write(pipe_fd[1],w_buf,1024))==1)
printf("write to pipe error ");
else
printf("the bytes write to pipe is %d ", writenum);
writenum=write(pipe_fd[1],w_buf,4096);
close(pipe_fd[1]);
}
}
}

```

输出结果：

```

the bytes write to pipe 1000
the bytes write to pipe 1000 //注意，此行输出说明了写入的非原子性
the bytes write to pipe 1000
the bytes write to pipe 1000
the bytes write to pipe 1000
the bytes write to pipe 120 //注意，此行输出说明了写入的非原子性
the bytes write to pipe 0
the bytes write to pipe 0
.....

```

结论：

写入数目小于4096时写入是非原子的！

如果把父进程中的两次写入字节数都改为5000，则很容易得出下面结论：

写入管道的数据量大于4096字节时，缓冲区的空闲空间将被写入数据（补齐），直到写完所有数据为止，如果没有进程读数据，则一直阻塞。

## 管道应用实例

### 实例一：用于 shell

管道可用于输入输出重定向，它将一个命令的输出直接定向到另一个命令的输入。比如，当在某个 shell 程序（Bourne shell 或 C shell 等）键入 `who | wc -l` 后，相应 shell 程序将创建 `who` 以及 `wc` 两个进程和这两个进程间的管道。考虑下面的命令行：

`$kill -l` 运行结果见 附一。

`$kill -l | grep SIGRTMIN` 运行  
30) SIGPWR 31) SIGSYS 32)



qzwujiaying

关注

10



17

```

34) SIGRTMIN+2 35) SIGRTMIN+3 36) SIGRTMIN+4 37)
SIGRTMIN+5
38) SIGRTMIN+6 39) SIGRTMIN+7 40) SIGRTMIN+8 41)
SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45)
SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49)
SIGRTMAX-14

```

### 实例二：用于具有亲缘关系的进程间通信

下面例子给出了管道的具体应用，父进程通过管道发送一些命令给子进程，子进程解析命令，并根据命令作相应处理。

```

#include <unistd.h>
#include <sys/types.h>
main()
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[4];
    char** w_buf[256];
    int chldexit=0;
    int i;
    int cmd;
    memset(r_buf,0,sizeof(r_buf));
    if(pipe(pipe_fd)<0)
    {
        printf("pipe create error ");
        return -1;
    }
    if((pid=fork())==0)
    //子进程：解析从管道中获取的命令，并作相应的处理
    {
        printf(" ");
        close(pipe_fd[1]);
        sleep(2);
        while(!chldexit)
        {
            read(pipe_fd[0],r_buf,4);
            cmd=atoi(r_buf);
            if(cmd==0)
            {
                printf("child: receive command from parent over now child process exit
");
                chldexit=1;
            }
            else if(handle_cmd(cmd)!=0)
                return;
            sleep(1);
        }
        close(pipe_fd[0]);
        exit();
    }
    else if(pid>0)
    //parent: send commands to child
    {
        close(pipe_fd[0]);
        w_buf[0]="003";
        w_buf[1]="005";
        w_buf[2]="777";
    }
}

```



qzwujiaying

关注

10



17

```
w_buf[3]="000";
for(i=0;i<4;i++)
write(pipe_fd[1],w_buf[i],4);
close(pipe_fd[1]);
}
}
//下面是子进程的命令处理函数（特定于应用）：
int handle_cmd(int cmd)
{
if((cmd<0)|| (cmd>256))
//suppose child only support 256 commands
{
printf("child: invalid command ");
return -1;
}
printf("child: the cmd from parent is %d ", cmd);
return 0;
}
```

## 管道的局限性

管道的 主要局限性正体现在它的特点上：

只支持单向数据流；只能用于具有亲缘关系的进程之间；没有名字；管道的缓冲区是有限的（管道制存在于内存中，在管道创建时，为缓冲区分配一个页面大小）；管道所传送的是无格式字节流，这就要求管道的读出方和写入方必须事先约定好数据的格式，比如多少字节算作一个消息（或命令、或记录）等等；

### Linux 管道的实现机制

在Linux中，管道是一种使用非常频繁的通信机制。从本质上说，管道也是一种文件，但它又和一般的文件有所不同，管道可以克服使用文件进行通信的两个问题，具体表现为：

限制管道的大小。实际上，管道是一个固定大小的缓冲区。在Linux中，该缓冲区的大小为1 页，即4K字节，使得它的大小不象文件那样不加检验地增长。使用单个固定缓冲区也会带来问题，比如在写管道时可能变满，当这种情况发生时，随后对管道的 write()调用将默认地被阻塞，等待某些数据被读取，以便腾出足够的空间供write()调用写。

读取进程也可能工作得比写进程快。当所有当前进程数据已被读取时，管道变空。当这种情况发生时，一个随后的read()调用将默认地被阻塞，等待某些数据被写入，这解决了read()调用返回文件结束的问题。

**注意：**从管道读数据是一次性操作，数据一旦被读，它就从管道中被抛弃，释放空间 以便写更多的数据。

### 1. 管道的结构

在 Linux 中，管道的实现并没有使用专门的数据结构，而是借助了文件系统的file结构和VFS的索引节点inode。通过将两个 file 结构指向同一个临时的 VFS 索引节点，而这个 VFS 索引节点又指向一个物理页面而实现的。如图 7.1所示。

图7.1 管道结构示意图

图7.1中有两个 file 数据结构，但它们定义文件操作例程地址是不同的，其中一个是指向管道中写入数据的例程地址，而另一个是从管道中读出数据的例程地址。这样，用户程序的 **系统调用** 仍然是 通常的文件操作，而内核却利用这种 **抽象机制** 实现了管道这一特殊操作。

### 2. 管道的读写

管道实现的源代码在fs/pipe.c中，在pipe.c中有很多函数，其中有两个函数比较重要，即管道读函数pipe\_read()和管道写函数pipe\_wrtie()。管道写函数通过将字节复制到 VFS 索引节点指向的物理内存而写入数据，而管道读函数则通过复制物理内存到用户空间而读出数据。管道利用一定的机制同步对管道的读写。



qzwujiaying

关注

10

17

17

信号。

当写进程向管道中写入时，它利用标准的库函数write()，系统根据库函数传递的文件描述符，可找到该文件的file结构。file结构中指定了用来进行写操作的函数（即写入函数）地址，于是，内核调用该函数完成写操作。写入函数在向内存中写入数据之前，必须首先检查VFS索引节点中的信息，同时满足如下条件时，才能进行实际的内存复制工作：

内存中有足够的空间可容纳所有要写入的数据；

内存没有被读程序锁定。

如果同时满足上述条件，写入函数首先锁定内存，然后从写进程的地址空间中复制数据到内存。否则，写入进程就休眠在VFS索引节点的等待队列中，接下来，内核将调用调度程序，而调度程序会选择其他进程运行。写入进程实际处于可中断的等待状态，当内存中有足够的空间可以容纳写入数据，或内存被解锁时，读取进程会唤醒写入进程，这时，写入进程将接收到信号。当数据写入内存之后，内存被解锁，而所有休眠在索引节点的读取进程会被唤醒。

管道的读取过程和写入过程类似。但是，进程可以在没有数据或内存被锁定时立即返回错误信息，而不是阻塞该进程，这依赖于文件或管道的打开模式。反之，进程可以休眠在索引节点的等待队列中等待写入进程写入数据。当所有的进程完成了管道操作之后，管道的索引节点被丢弃，而共享数据页也被释放。

因为管道的实现涉及很多文件的操作,因此,当读者学完有关文件系统的内容后来读pipe.c中的代码，你会觉得并不难理解。

来自：  
<http://hi.baidu.com/jrckkyy/blog/item/dbbb63098ae0e08ed0581b4a.html>

文章知识点与官方知识档案匹配，可进一步学习相关知识

算法技能树 首页 概览 36885 人正在系统学习中

- pipe.c文件

管道通信中的实验，pipe.c文件，下载后直接编译即可

06-11
- C例子：IPC-pipe3

该程序是我写的博客“一起talk C栗子吧（第九十回：C语言实例--使用管道进行...”

01-04
- C语言函数声明以及函数原型 最新发布

C语言函数声明以及函数原型

Elanie1024的博客 500
- Linux C++ 进程间通信-PIPE

【代码】Linux C++ 进程间通信-PIPE。

u014630983的博客 222
- c语言pipe函数,pipe 函数 (C语言)

pipe我们用中文叫做管道。以下讲解均是基于Linux为环境：函数简介所需头文...

weixin\_35002851的博客 1659
- [C] pipe以及fork使用

进程与管道的使用 参考原文链接：https://blog.csdn.net/nodeathphoenix/article/...

afleshfish的博客 544
- C例子：IPC-pipe

该程序是我写的博客“一起talk C栗子吧（第八十八回：C语言实例--使用管道进...

01-03
- linux c之pipe的使用例子

#include /\* some systems still require this \*/ #include #include /\* for winsize \*/ #...

云守护的专栏 3691
- C语言 pipe

每个进程拥有各自的地址空间,一个进程的全局变量在另一个进程中看不到,所以...

Claroja 850
- 了解C语言中的pipe()系统调用

文章目录基本概念父子进程共享管道参考文档 基本概念 从概念上讲，管道...

zsx0728的博客 2176
- linux c语言之pipe () 函数 热门推荐

函数简介函数原型 int pipe(int fd[2]) 函数传入值 fd[2]管道的两个文件描述符

大圣编了个程 1万+



// 多进程以及ipc管道方式进程间通信 /\* 1、父进程调用pipe开辟管道，得到两个...

linuxC多进程通讯---pipe同步 qq\_23929673的博客 362  
文章目录例子解析 例子 #include <stdio.h> #include <stdlib.h> #include <unistd....

Linux C pipe函数 man 翻译 矩阵实验室 1144  
Linux C pipe函数 man 翻译

linux pipe.c 源代码,linux 下PIPE管道通信... weixin\_32999557的博客 197  
学习linux 已经 有一段时间了。把之前学习中一些东西记下来，和大家分享。/\*\*...

代码超详细讲解:C语言进程间通信之SuperMario管道... HenryBlog 388  
client和server之间如何通信举个简单例子： client端标准输入standard input，读...

C 语言编程 — 管道 (Pipe) 烟云的计算 6562  
目录 文章目录目录Linux 的管道指令C 语言的匿名管道命名管道匿名管道和命名...

linux c pipe乱码,c – 在Linux中读写PIPE weixin\_34570241的博客 162  
linux中一个简单的多进程程序.输入一些数字,如./findPrime 10 20 30.该程序将创...

C语言读取pipe缓冲区内容 SKJavaCsdn的博客 501  
#include<stdio.h> int main() { int x, fd[2]; char buf[30],s[30]; pipe(fd); while((x = ...

“相关推荐”对你有帮助么？

- 非常没帮助
- 没帮助
- 一般
- 有帮助
- 非常有帮助

©2022 CSDN 皮肤主题：大白 设计师：CSDN官方博客 返回首页

关于 招贤 商务 寻求 400-  
我们 纳士 合作 报道 660-  
0108  
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息  
北京互联网违法和不良信息举报中心 家长监护 网络110报警服务 中国互联网举报中心  
Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照  
©1999-2023北京创新乐知网络技术有限公司



qzwujiaying  
码龄15年 暂无认证

35 23万+ 196万+ 27万+   
原创 周排名 总排名 访问 等级

2048 21 15 2 30  
积分 粉丝 获赞 评论 收藏



私信

关注

搜博文文章



热门文章

- shell--传入参数的处理 138288
- pipe 函数 (C语言) 17529
- AIX上ulimit -a 输出的含义 10429
- vsftpd 安装(vsftpd 530 login incorrect 报错)  
9110
- Zh\_CN.GB18030 判断是否为中文 6310





qzwujiaying

关注

10 17



	C语言 (linux/unix/AIX)	14篇
	linux/unix/AIX 系统方面	4篇
	Makefile符号说明	1篇
	Oracle数据库/数据库知识	8篇
	perl语言	2篇



最新评论

shell--传入参数的处理  
夏吃萝卜: 有用，不错，谢谢  
Linux/AIX/UNIX 大文件排序问题  
ccdn2022: 您好，我这两天也在处理一个3.2G的大文件，要对其第一个字段进行排 ...

您愿意向朋友推荐“博客详情页”吗？



强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

EditPlus 快捷键  
金额字段加千位分隔符  
ORA-00980: synonym translation is no longer valid", version 9  
2012年 1篇      2011年 27篇  
2010年 26篇

目录

- 函数简介
- 什么是管道
- 管道的创建
- 管道的读写规则
- 管道应用实例
- 管道的局限性