

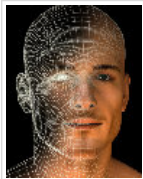


zxg623

zxg623.blog.chinaunix.net

只为伊人守候 和我共同营造苍凉的远方 醉解千愁，他处不堪留 我冷因为我看到世界的冷漠，我做因为孤独的灵魂漂泊于芸芸众生之上，我酷因为没被事故磨去棱角，我狂因为愿意用生命去追求理想，我痴因为还相信爱情的魔力 我的笑隐约透露着孤独，快乐背后深深藏着痛楚，坚强的面对然后偷偷的哭.....

首页 | 博文目录 | 关于我



zxg623

博客访问： 953200
博文数量： 273
博客积分： 10418
博客等级： 上将
技术积分： 3308
用 户 组： 普通用户
注册时间： 2008-04-09 23:49

[加关注](#) [短消息](#)
[论坛](#) [加好友](#)

文章分类

- 全部博文 (273)
- android (3)
- ubuntu (7)
- MTK (5)
- 网络协议 (7)
- windows技术 (19)
- 网络技术 (4)
- 硬件技术 (1)
- 技术手册 (3)
- 杂文 (16)
- 操作系统 (3)
- c++ (10)
- c (41)
- 嵌入式 (45)
- linux (98)
- 未分配的博文 (11)

文章存档

- 2014年 (1)
- 2013年 (5)
- 2012年 (11)
- 2011年 (2)
- 2010年 (8)
- 2009年 (28)
- 2008年 (218)
- 2008年12月 (9)

Linux 上实现双向进程间通信管道

2008-09-28 08:42:34

分类： LINUX

本文阐述了一个使用 socketpair 系统调用在 Linux 上实现双向进程通讯管道的方法，并提供了一个实现。

问题和常见方法

Linux 提供了 popen 和 pclose 函数 (1)，用于创建和关闭管道与另外一个进程进行通信。其接口如下：

```
FILE *popen(const char *command, const char *mode);  
int pclose(FILE *stream);
```

遗憾的是，popen 创建的管道只能是单向的 -- mode 只能是 “r” 或 “w” 而不能是某种组合--用户只能选择要么往里写，要么从中读，而不能同时在一个管道中进行读写。实际应用中，经常会有同时进行读写的要求，比如，我们可能希望把文本数据送往sort工具排序后再取回结果。此时popen就无法用上了。我们需要寻找其它的解决方案。

有一种解决方案是使用 pipe 函数 (2)创建两个单向管道。没有错误检测的代码示意如下：

```
int pipe_in[2], pipe_out[2];  
pid_t pid;  
pipe(&pipe_in); // 创建父进程中用于读取数据的管道  
pipe(&pipe_out); // 创建父进程中用于写入数据的管道  
if ( (pid = fork()) == 0) { // 子进程  
    close(pipe_in[0]); // 关闭父进程的读管道的子进程读端  
    close(pipe_out[1]); // 关闭父进程的写管道的子进程写端  
    dup2(pipe_in[1], STDOUT_FILENO); // 复制父进程的读管道到子进程的标准输出  
    dup2(pipe_out[0], STDIN_FILENO); // 复制父进程的写管道到子进程的标准输入  
    close(pipe_in[1]); // 关闭已复制的读管道  
    close(pipe_out[0]); // 关闭已复制的写管道  
    /* 使用exec执行命令 */  
} else { // 父进程  
    close(pipe_in[1]); // 关闭读管道的写端  
    close(pipe_out[0]); // 关闭写管道的读端  
    /* 现在可向pipe_out[1]中写数据，并从pipe_in[0]中读结果 */  
    close(pipe_out[1]); // 关闭写管道  
    /* 读取pipe_in[0]中的剩余数据 */  
    close(pipe_in[0]); // 关闭读管道  
    /* 使用wait系列函数等待子进程退出并取得退出代码 */  
}
```

当然，这样的代码的可读性（特别是加上错误处理代码之后）比较差，也不容易封装成类似于 popen/pclose的函数，方便高层代码使用。究其原因，是pipe函数返回的一对文件描述符只能从第一个中读、第二个中写（至少对于Linux是如此）。为了同时读写，就只能采取这么累赘的两个pipe调用、两个文件描述符的形式了。

2008年11月（6）

2008年10月（12）

2008年09月（7）

2008年08月（3）


2008年07月（10）

2008年06月（13）


2008年05月（16）

2008年04月（142）


我的朋友




程睿




huhuawang




pzm0729




gongping




gududesi



小雅贝贝




wbshwxn




52dreame


最近访客




邈宝宝




welt7




lxlxt




thu寒枫




qian1987




patrickb



64492407




蜗牛ai小




zjhunan1

微信关注



IT168企业级官微
微信号：IT168qiye



系统架构师大会
微信号：SACC2013

订阅

推荐博文

• 淘宝分布式文件系统TFS设计...

• Kerberos 服务的工作原理...

• RHEL7下配置Kerberos+LDAP+NF...

• TortoiseGit putty key

• docker的跨主机解决方案weave...

热词专题

• lua编译(linux)

一个更好的方案

使用pipe就只能如此了。不过，Linux实现了一个源自BSD的socketpair调用（3），可以实现上述在同一个文件描述符中进行读写的功能（该调用目前也是POSIX规范的一部分（4））。该系统调用能创建一对已连接的（UNIX族）无名socket。在Linux中，完全可以把这一对socket当成pipe返回的文件描述符一样使用，唯一的区别就是这一对文件描述符中的任何一个都可读和可写。

这似乎可以是一个用来实现进程间通信管道的好方法。不过，要注意的是，为了解决我前面的提出的使用sort的应用问题，我们需要关闭子进程的标准输入通知子进程数据已经发送完毕，而后从子进程的标准输出中读取数据直到遇到EOF。使用两个单向管道的话每个管道可以单独关闭，因而不存在任何问题；而在使用双向管道时，如果不关闭管道就无法通知对端数据已经发送完毕，但关闭了管道又无法从中读取结果数据。——这一问题不解决的话，使用socketpair的设想就变得毫无意义。

令人高兴的是，shutdown调用（5）可解决此问题。毕竟socketpair产生的文件描述符是一对socket，socket上的标准操作都可以使用，其中也包括shutdown。——利用shutdown，可以实现一个半关闭操作，通知对端本进程不再发送数据，同时仍可以利用该文件描述符接收来自对端的数据。没有错误检测的代码示意如下：

```
int fd[2];
pid_t pid;
socketpair(AF_UNIX, SOCKET_STREAM, 0, fd); // 创建管道
if ( (pid = fork()) == 0 ) { // 子进程
    close(fd[0]); // 关闭管道的父进程端
    dup2(fd[1], STDOUT_FILENO); // 复制管道的子进程端到标准输出
    dup2(fd[1], STDIN_FILENO); // 复制管道的子进程端到标准输入
    close(fd[1]); // 关闭已复制的读管道
    /* 使用exec执行命令 */
} else { // 父进程
    close(fd[1]); // 关闭管道的子进程端
    /* 现在可在fd[0]中读写数据 */
    shutdown(fd[0], SHUT_WR); // 通知对端数据发送完毕
    /* 读取剩余数据 */
    close(fd[0]); // 关闭管道
    /* 使用wait系列函数等待子进程退出并取得退出代码 */
}
```

很清楚，这比使用两个单向管道的方案要简洁不少。我将在此基础上作进一步的封装和改进。

封装和实现

直接使用上面的方法，无论怎么看，至少也是丑陋和不方便的。程序的维护者想看到的是程序的逻辑，而不是完成一件任务的各种各样的繁琐细节。我们需要一个好的封装。

封装可以使用C或者C++。此处，我按照UNIX的传统，提供一个类似于POSIX标准中popen/pclose函数调用的C封装，以保证最大程度的可用性。接口如下：

```
FILE *dpopen(const char *command);
int dpclose(FILE *stream);
int dphalfclose(FILE *stream);
```

关于接口，以下几点需要注意一下：

- 与pipe函数类似，dpopen返回的是文件结构的指针，而不是文件描述符。这意味着，我们可以直接使用fprintf之类的函数，文件缓冲区会缓存写入管道的数据（除非使用setbuf函数关闭文件缓冲区），要保证数据确实写入到管道中需要使用fflush函数。
- 由于dpopen返回的是可读写的管道，所以popen的第二个表示读/写的参数不再需要。

- 在双向管道中我们需要通知对端写数据已经结束，此项操作由dphalfclose函数来完成。
具体的实现请直接查看程序源代码，其中有详细的注释和doxygen文档注释 (6)。我只略作几点说明：
- 本实现使用了一个链表来记录所有dopen打开的文件指针和子进程ID的对应关系，因此，在同时用dopen打开的管道的多的时候，dpclose（需要搜索链表）的速度会稍慢一点。我认为在通常使用过程中这不会产生什么问题。如果在某些特殊情况下这会是一个问题的话，可考虑更改dopen的返回值类型和dpclose的传入参数类型（不太方便使用，但实现简单），或者使用哈希表/平衡树来代替目前使用的链表以加速查找（接口不变，但实现较复杂）。
- 当编译时在gcc中使用了“-pthread”命令行参数时，本实现会启用POSIX线程支持，使用互斥量保护对链表的访问。因此本实现可以安全地用于POSIX多线程环境之中。
- 与popen类似 (7)，dopen会在fork产生的子进程中关闭以前用dopen打开的管道。
- 如果传给dpclose的参数不是以前用dopen返回的非NULL值，当前实现除返回-1表示错误外，还会把errno设为EBADF。对于pclose而言，这种情况在POSIX规范中被视为不确定（unspecified）行为 (8)。
- 实现中没有使用任何平台相关特性，以方便移植到其它POSIX平台上。

下面的代码展示了一个简单例子，将多行文本送到sort中，然后取回结果、显示出来：

```
#include
#include
#include "dopen.h"
#define MAXLINE 80
int main()
{
    char    line[MAXLINE];
    FILE    *fp;
    fp = dopen("sort");
    if (fp == NULL) {
        perror("dopen error");
        exit(1);
    }
    fprintf(fp, "orange\n");
    fprintf(fp, "apple\n");
    fprintf(fp, "pear\n");
    if (dphalfclose(fp) < 0) {
        perror("dphalfclose error");
        exit(1);
    }
    for (;;) {
        if (fgets(line, MAXLINE, fp) == NULL)
            break;
        fputs(line, stdout);
    }
    dpclose(fp);
    return 0;
}
```

输出结果为：

```
apple
orange
pear
```

[➤ 回首页](#)

总结

本文阐述了一个使用socketpair系统调用在Linux上实现双向进程通讯管道的方法，并提供了一个实现。该实现提供的接口与POSIX规范中的popen/pclose函数较为接近，因而非常易于使用。该实现没有使用平

台相关的特性，因而可以不加修改或只进行少量修改即可移植到支持socketpair调用的POSIX系统中去。
本文源码下载：[dpopen.zip](#)

参考资料

- 1. 相应的man (3)页。在线查看：<http://www.die.net/doc/linux/man/man3/popen.3.html>
- 2. 相应的man (2)页。在线查看：<http://www.die.net/doc/linux/man/man2/pipe.2.html>
- 3. 相应的man (2)页。在线查看：<http://www.die.net/doc/linux/man/man2/socketpair.2.html>
- 4. POSIX规范：<http://www.opengroup.org/onlinepubs/009695399/functions/socketpair.html>
- 5. 相应的man (2)页。在线查看：<http://www.die.net/doc/linux/man/man2/shutdown.2.html>
- 6. Doxygen主页：<http://www.stack.nl/~dimitri/doxygen/>
- 7. POSIX规范：<http://www.opengroup.org/onlinepubs/009695399/functions/popen.html>
- 8. POSIX规范：<http://www.opengroup.org/onlinepubs/009695399/functions/pclose.html>

阅读 (3968) | 评论 (0) | 转发 (1) |

上一篇：[如何实现自动登录Linux](#)
下一篇：[Linux环境进程间通信（一）：管道及有名管道](#)

0

相关热门文章

linux 常见服务端口	linux dhcp peizhi roc
xmanager 2.0 for linux配置	关于Unix文件的软链接
【ROOTFS搭建】busybox的httpd...	求教这个命令什么意思，我是新...
openwrt中luci学习笔记	sed -e "/grep/d" 是什么意思...
什么是shell	谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧！~~

评论热议

登录后评论。
[登录](#) [注册](#)