Linux IPC 之管道

1.2. 标准流管道

像文件操作有标准 io 流一样，管道也支持文件流模式。用来创建连接到另一进程的管道，是通过函数 popen 和 pclose。

函数原型：

```
#include <stdio.h>
FILE* popen(const char* command, const char* open_mode);
int pclose(FILE* fp);
```

函数 popen()：允许一个程序将另一个程序作为新进程来启动，并可以传递数据给它或者通过它接收数据。command 字符串是要运行的程序名。open_mode 必须是“r”或“w”。如果 open_mode 是“r”，被调用程序的输出就可以被调用程序(popen)使用，调用程序利用 popen 函数返回的 FILE*文件流指针，就可以通过常用的 stdio 库函数（如 fread）来读取被调用程序的输出；如果 open_mode 是“w”，调用程序(popen)就可以用 fwrite 向被调用程序发送数据，而被调用程序可以在自己的标准输入上读取这些数据。

函数 pclose()：用 popen 启动的进程结束时，我们可以用 pclose 函数关闭与之关联的文件流。

Example1：从标准管道流中读取　打印/etc/profile 的内容

```
#include <stdio.h>
int main()
{
    FILE* fp = popen("cat /etc/profile", "r");
    char buf[512] = {0};
    while(fgets(buf, sizeof(buf), fp))
    {
        puts(buf);
    }
    pclose(fp);
    return 0;
}
```

Example2：写到标准管道流　　统计 buf 单词数(被调用程序必须阻塞等待标准输入)

```
#include<stdio.h>
int main()
{
    char buf[]={"aaa  bbb  ccc  ddd  eee  fff  ggg  hhh"};
    FILE *fp = popen("wc -w", "w");
    fwrite(buf, sizeof(buf), 1, fp);
    pclose(fp);
    return 0;
}
```

1.3. 无名管道(PIPE)

管道是 linux 进程间通信的一种方式，如命令 ps -ef | grep ntp

无名管道的特点：

1 只能在亲缘关系进程间通信（父子或兄弟）

2 半双工（固定的读端和固定的写端）

3 他是特殊的文件，可以用 read、write 等，只能在内存中

管道函数原型：

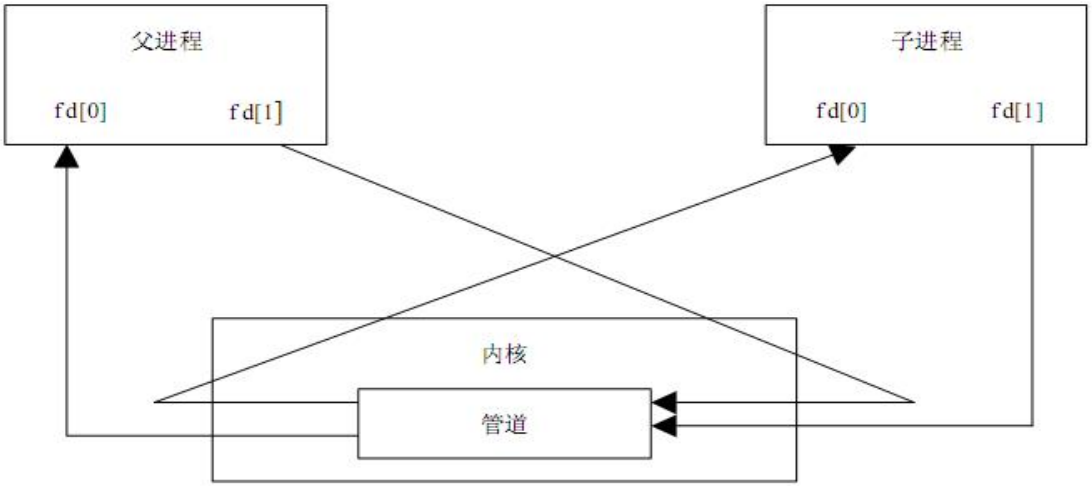
```
#include <unistd.h>
int pipe(int fds[2]);
```

管道在程序中用一对文件描述符表示，其中一个文件描述符有可读属性，一个有可写的属性。fds[0]是读，fds[1]是写。

函数 pipe 用于创建一个无名管道，如果成功，fds[0]存放可读的文件描述符，fds[1]存放可写文件描述符,并且函数返回 0，否则返回-1。

通过调用 pipe 获取这对打开的文件描述符后，一个进程就可以从 fds[0]中读数据，而另一个进程就可以往 fds[1]中写数据。当然两进程间必须有继承关系，才能继承这对打开的文件描述符。

管道不象真正的物理文件，不是持久的，即两进程终止后，管道也自动消失了。



示例：创建父子进程，创建无名管道，父写子读

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int main()
{
    int fds[2] = {0};
    pipe(fds);
    char szBuf[32] = {'\0'};
    if(fork() == 0){      //表示子进程
        close(fds[1]);  //子进程关闭写
        sleep(2);       //确保父进程有时间关闭读，并且往管道中写内容
        if(read(fds[0], szBuf, sizeof(szBuf)) > 0)
            puts(buf);
```

```
            close(fds[0]);  //子关闭读
            exit(0);
        }else{                  //表示父进程
            close(fds[0]);  //父关闭读
            write(fds[1],    "hello", 6);
            waitpid(-1, NULL, 0);          //等子关闭读
            //write(fds[1], "world",6);      //此时将会出现“断开的管道”因为子的读已经关闭
    了
            close(fds[1]);  //父关闭写
            exit(0);
        }
        return 0;
    }
```

管道两端的关闭是有先后顺序的，如果先关闭写端则从另一端读数据时，read 函数将返回 0，表示管道已经关闭；但是如果先关闭读端，则从另一端写数据时，将会使写数据的进程接收到 SIGPIPE 信号，如果写进程不对该信号进行处理，将导致写进程终止，如果写进程处理了该信号，则写数据的 write 函数返回一个负值，表示管道已经关闭。示例：

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int main()
{
    int fds[2];
    pipe(fds);
    //注释掉这部分将导致写进程被信号SIGPIPE终止
    sigset_t setSig;
    sigemptyset(&setSig);
    sigaddset(&setSig,SIGPIPE);
    sigprocmask(SIG_BLOCK,&setSig,NULL);
    char szBuf[10] = {0};
    if(fork() == 0){
        close(fds[1]);//子关闭写
        sleep(2);//确保父关闭读
        if(read(fds[0], szBuf, sizeof(szBuf)) > 0)
            puts(szBuf);
            close(fds[0]);//子关闭读
    }else{
        close(fds[0]);//父关闭读
        write(fds[1], "hello", 6);
        wait(NULL);
        write(fds[1], "world", 6);//子的读关闭，父还在写
        close(fds[1]);//父关闭写
    }
```

```
    return 0;
}
```

1.4. 命名管道(FIFO)

无名管道只能在亲缘关系的进程间通信大大限制了管道的使用，有名管道突破了这个限制，通过指定路径名的范式实现不相关进程间的通信

1.4.1. 创建、删除 FIFO 文件

创建 FIFO 文件与创建普通文件很类似，只是创建后的文件用于 FIFO。

1. 用函数创建和删除 FIFO 文件

• 创建 FIFO 文件的函数原型：

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

参数 pathname 为要创建的 FIFO 文件的全路径名；

参数 mode 为文件访问权限

如果创建成功，则返回 0，否则-1。

示例：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc,char *argv[])//演示通过命令行传递参数
{
    if(argc != 2){
        puts("Usage: MkFifo.exe {filename}");
        return -1;
    }
    if(mkfifo(argv[1], 0666) == -1){
        perror("mkfifo fail");
        return -2;
    }
    return 0;
}
```

• 删除FIFO文件的函数原型为：

```
#include <unistd.h>
    int unlink(const char *pathname);
```

示例：

```
#include <unistd.h>
main()
{
```

```
        unlink("pp");
    }
```

2. 用命令创建和删除FIFO文件

- 用命令mkfifo创建　不能重复创建
- 用命令unlink删除

创建完毕之后，就可以访问FIFO文件了：

一个终端：cat < myfifo

另一个终端：echo “hello” > myfifo

1.4.2. 打开、关闭 FIFO 文件

对 FIFO 类型的文件的打开/关闭跟普通文件一样，都是使用 open 和 close 函数。如果打开时使用 O_WRONLY 选项，则打开 FIFO 的写入端，如果使用 O_RDONLY 选项，则打开 FIFO 的读取端，写入端和读取端都可以被几个进程同时打开。

如果以读取方式打开 FIFO，并且还没有其它进程以写入方式打开 FIFO，open 函数将被阻塞；同样，如果以写入方式打开 FIFO，并且还没其它进程以读取方式 FIFO，open 函数也将被阻塞。

与 PIPE 相同，关闭 FIFO 时，如果先关读取端，将导致继续往 FIFO 中写数据的进程接收 SIGPIPE 的信号。

1.4.3. 读写 FIFO

可以采用与普通文件相同的读写方式读写 FIFO,

Example：先执行#mkfifo MyFifo.pip 命令

然后 vi write.c 如下：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
int main()
{
    int fdFifo = open("MyFifo.pip",O_WRONLY);   //1. 打开（判断是否成功打开略）
    write(fdFifo, "hello", 6);                  //2. 写
    close(fdFifo);                              //3. 关闭
    return 0;
}
```

然后 vi read.c 如下：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
int main()
{
```

```
    char szBuf[128];
    int fdFifo = open("MyFifo.pip",O_RDONLY);    //1. 打开
    if(read(fdFifo,szBuf,sizeof(szBuf)) > 0)     //2. 读
        puts(szBuf);
    close(fdFifo);                               //3. 关闭
    return 0;
}
```

然后　gcc –o write write.c

gcc –o read read.c

./write　//发现阻塞，要等待执行./read

./read

在屏幕上输出 hello

管道示例：基于管道的客户端服务器程序。

程序说明：

1. 服务器端：

   维护服务器管道，接受来自客户端的请求并处理（本程序为接受客户端发来的字符串，将小写字母转换为大写字母。）然后通过每个客户端维护的管道发给客户端。

2. 客户端

   向服务端管道发送数据，然后从自己的客户端管道中接受服务器返回的数据。

   示例代码见下：

   服务器端 server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
typedef struct tagmag
{
        int client_pid ;
        char my_data[512] ;
}MSG;
int main()
{
        int server_fifo_fd , client_fifo_fd ;
        char client_fifo[256];
        MSG my_msg ;
        char * pstr ;
        memset(&my_msg , 0 , sizeof(MSG));
        mkfifo("SERVER_FIFO_NAME",0777);
        server_fifo_fd = open("./SERVER_FIFO_NAME",O_RDONLY);
    if(server_fifo_fd == -1)
```

```
        {
                        perror("server_fifo_fd");
                        exit(-1);
                }
                int iret ;
                while(    (iret = read(server_fifo_fd , &my_msg ,sizeof(MSG))>0))
                {
                //iret = read(server_fifo_fd , &my_msg ,sizeof(MSG));
                          pstr = my_msg.my_data ;
                        printf("%s\n",my_msg.my_data);
                        while(*pstr != '\0')
                        {
                                *pstr = toupper(*pstr);
                                pstr ++ ;
                        }
                        memset(client_fifo , 0 , 256);
                        sprintf(client_fifo , "CLIENT_FIFO_%d" , my_msg.client_pid);
                        client_fifo_fd = open(client_fifo , O_WRONLY);
                        if(client_fifo_fd == -1)
                        {
                        perror("client_fifo_fd");
                        exit(-1);
                    }
        write(client_fifo_fd , &my_msg, sizeof(MSG));
                        printf("%s\n", my_msg.my_data);
                        printf("OVER!\n");
                        close(client_fifo_fd);

                }
                return 0 ;
}
```

客户端代码：client.c

```
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        #include <sys/types.h>
        #include <sys/stat.h>
        #include <unistd.h>
        #include <fcntl.h>
        typedef struct tagmag
        {
                int client_pid ;
                char my_data[512] ;
        }MSG;
```

```
int main()
{
        int server_fifo_fd,client_fifo_fd ;
        char client_fifo[256]={0};
        sprintf(client_fifo,"CLIENT_FIFO_%d",getpid());
        MSG my_msg ;
        memset(&my_msg , 0 , sizeof(MSG));
        my_msg.client_pid = getpid();
        server_fifo_fd = open("./SERVER_FIFO_NAME",O_WRONLY);
        mkfifo(client_fifo , 0777);
        while(1)
 {
 int n = read(STDIN_FILENO,my_msg.my_data , 512);
 my_msg.my_data[n] = '\0' ;
 write(server_fifo_fd , &my_msg , sizeof(MSG));
 client_fifo_fd = open(client_fifo , O_RDONLY);
 //memset(&my_msg , 0    , sizeof(MSG));
 n = read(client_fifo_fd,&my_msg , sizeof(MSG));
 my_msg.my_data[n]= 0 ;
 write(STDOUT_FILENO, my_msg.my_data ,strlen(my_msg.my_data) );
 close(client_fifo_fd);

 }
 unlink(client_fifo);
}
```