[Linux进程间通信——使用命名管道](http://blog.csdn.net/ljianhui/article/details/10202699)

一、什么是命名管道

命名管道也被称为FIFO文件，它是一种特殊类型的文件，它在文件系统中以文件名的形式存在，但是它的行为却和之前所讲的没有名字的管道（匿名管道）类似。

由 于Linux中所有的事物都可被视为文件，所以对命名管道的使用也就变得与文件操作非常的统一，也使它的使用非常方便，同时我们也可以像平常的文件名一样在命令中使用。

二、创建命名管道

我们可以使用两下函数之一来创建一个命名管道，他们的原型如下：

1. #include <sys/types.h>
2. #include <sys/stat.h>
3. **int** mkfifo(**const** **char** \*filename, mode\_t mode);
4. **int** mknod(**const** **char** \*filename, mode\_t mode | S\_IFIFO, (dev\_t)0);

这两个函数都能创建一个FIFO文件，注意是创建一个真实存在于文件系统中的文件，filename指定了文件名，而mode则指定了文件的读写权限。

mknod是比较老的函数，而使用mkfifo函数更加简单和规范，所以建议在可能的情况下，尽量使用mkfifo而不是mknod。

三、访问命名管道

1、打开FIFO文件

与打开其他文件一样，FIFO文件也可以**使用open调用来打开**。注意，mkfifo函数只是创建一个FIFO文件，要使用命名管道还是将其打开。

但是有两点要注意，1、就是程序不能以O\_RDWR模式打开FIFO文件进行读写操作，而其行为也未明确定义，因为如一个管道以读/写方式打开，进程就会读回自己的输出，同时我们通常使用FIFO只是为了单向的数据传递。2、就是传递给open调用的是FIFO的路径名，而不是正常的文件。

打开FIFO文件通常有四种方式，

1. open(**const** **char** \*path, O\_RDONLY); //1
2. open(**const** **char** \*path, O\_RDONLY | O\_NONBLOCK);//2
3. open(**const** **char** \*path, O\_WRONLY); //3
4. open(**const** **char** \*path, O\_WRONLY | O\_NONBLOCK);//4

在open函数的调用的第二个参数中，你看到一个陌生的选项O\_NONBLOCK，选项O\_NONBLOCK表示非阻塞，加上这个选项后，表示open调用是非阻塞的，如果没有这个选项，则表示open调用是阻塞的。

open调用的阻塞是什么一回事呢？很简单，对于以只读方式（O\_RDONLY）打开的FIFO文件，如果open调用是阻塞的（即第二个参数为O\_RDONLY），除非有一个进程以写方式打开同一个FIFO，否则它不会返回；如果open调用是非阻塞的的（即第二个参数为O\_RDONLY | O\_NONBLOCK），则即使没有其他进程以写方式打开同一个FIFO文件，open调用将成功并立即返回。

对于以只写方式（O\_WRONLY）打开的FIFO文件，如果open调用是阻塞的（即第二个参数为O\_WRONLY），open调用将被阻塞，直到有一个进程以只读方式打开同一个FIFO文件为止；如果open调用是非阻塞的（即第二个参数为O\_WRONLY | O\_NONBLOCK），open总会立即返回，但如果没有其他进程以只读方式打开同一个FIFO文件，open调用将返回-1，并且FIFO也不会被打开。

四、命名管道的安全问题

前面的例子是两个进程之间的通信问题，也就是说，一个进程向FIFO文件写数据，而另一个进程则在FIFO文件中读取数据。试想这样一个问题，只使用一个FIFO文件，如果有多个进程同时向同一个FIFO文件写数据，而只有一个读FIFO进程在同一个FIFO文件中读取数据时，会发生怎么样的情况呢，会发生数据块的相互交错是很正常的？

为了解决这一问题，**就是让写操作的原子化**。怎样才能使写操作原子化呢？答案很简单，系统规定：在一个以O\_WRONLY（即阻塞方式）打开的FIFO中， 如果写入的数据长度小于等待PIPE\_BUF，那么或者写入全部字节，或者一个字节都不写入。如果所有的写请求都是发往一个阻塞的FIFO的，并且每个写记请求的数据长度小于等于PIPE\_BUF字节，系统就可以确保数据决不会交错在一起。

五、命名管道与匿名管道的对比

使用匿名管道，则通信的进程之间需要一个父子关系，通信的两个进程一定是由一个共同的祖先进程启动。但是匿名管道没有上面说到的数据交叉的问题。

与使用匿名管道相比，我们可以看到fifowrite.exe和fiforead.exe这两个进程是没有什么必然的联系的，如果硬要说他们具有某种联系，就只能说是它们都访问同一个FIFO文件。它解决了之前在匿名管道中出现的通信的两个进程一定是由一个共同的祖先进程启动的问题。但是为了数据的安全，我们很多时候要采用阻塞的FIFO，让写操作变成原子操作。