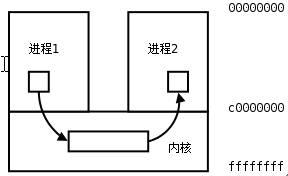
[进程间通信IPC之--无名管道（pipe）和有名管道（fifo）](http://blog.chinaunix.net/uid-25365622-id-3059840.html) 2012-01-17 22:41:20

分类： C/C++

每个进程各自有不同的用户地址空间，任何一个进 程的全局变量在另一个进程中都看不到，所以进程之间要交换数据必须通过内核，在内核中开辟一块缓冲 区，进程1把数据从用户空间拷到内核缓冲区，进程2再从内核缓冲区把数据读走，内核提供的这种机制称为进程间通信（IPC，InterProcess Communication）

如下图所示：

[](http://blog.chinaunix.net/attachment/201201/17/25365622_1326810872K240.jpg)

进程间通信共七种方式：

第一类：传统的unix通信机制：

# 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。  
# 有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

# 信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

第二类：System V IPC：   
# 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。  
# 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。  
# 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

第三类：BSD 套接字：

# 套接字( socket ) ： 套解字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

本文介绍IPC之无名管道（pipe）和有名管道（fifo）

**1、 管道概述及相关API应用**

**1.1 管道相关的关键概念**

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。

**1.2管道的创建：**

|  |
| --- |
| #include  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）。

**1.3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：（PIPE\_BUF在include/linux/limits.h中定义，不同的内核版本可能会有所不同。Posix.1要求PIPE\_BUF至少为512字节，red hat 7.2中为4096）。

向管道中写入数据：

* 向管道中写入数据时，linux将不保证写入的原子性，管道缓冲区一有空闲区域，写进程就会试图向管道写入数据。如果读进程不读走管道缓冲区中的数据，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SIFPIPE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）。

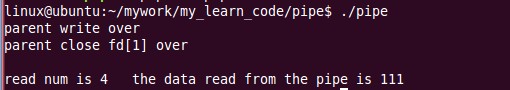
**1.4管道的局限性**

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；

无代码，无真相

1. /\*function:利用无名管道pipe实现有血缘关系的进程间通信，pipo\_fd[0]是读端，pipo\_fd[1]是写端。pipe是半双工的。
2. \* \*/
3. #include <unistd.h>
4. #include <sys/types.h>
5. #include <errno.h>
6. #include <stdlib.h>
7. #include <stdio.h>
8. #include <string.h>
9. int main()
10. {
11. int pipe\_fd[2];
12. pid\_t pid;
13. char r\_buf[10];
14. char w\_buf[4];
15. int r\_num;
16. memset(r\_buf,0,sizeof(r\_buf));
17. memset(w\_buf,0,sizeof(w\_buf));
18. if(pipe(pipe\_fd)<0)
19. {
20. printf("pipe create error\n");
21. return -1;
22. }
23. if((pid=fork())==0)
24. {
25. printf("\n");
26. close(pipe\_fd[1]);
27. sleep(3);//确保父进程关闭写端
28. r\_num=read(pipe\_fd[0],r\_buf,10);
29. printf(    "read num is %d the data read from the pipe is %d\n",r\_num,atoi(r\_buf));
30. close(pipe\_fd[0]);
31. exit(1);
32. }
33. else if(pid>0)
34. {
35. close(pipe\_fd[0]);//close read
36. strcpy(w\_buf,"111");
37. if(write(pipe\_fd[1],w\_buf,4)!=-1)
38. printf("parent write over\n");
39. printf("parent close fd[1] over\n");
40. close(pipe\_fd[1]);//write
41. sleep(10);
42. }
43. return 0;
44. }

[](http://blog.chinaunix.net/attachment/201201/17/25365622_13268131942vPt.jpg)

**2、 有名管道概述及相关API应用**

**2.1 有名管道相关的关键概念**

管道应用的一个重大限制是它没有名字，因此，只能用于具有亲缘关系的进程间通信，在有名管道（named pipe或FIFO）提出后，该限制得到了克服。FIFO不同于管道之处在于它提供一个路径名与之关联，以FIFO的文件形式存在于文件系统中。这样，即使与FIFO的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过FIFO相互通信（能够访问该路径的进程以及FIFO的创建进程之间），因此，通过FIFO不相关的进程也能交换数据。值得注意的是，FIFO严格遵循先进先出（first in first out），对管道及FIFO的读总是从开始处返回数据，对它们的写则把数据添加到末尾。它们不支持诸如lseek()等文件定位操作。

**2.2有名管道的创建**

|  |
| --- |
| #include  #include  int mkfifo(const char \* pathname, mode\_t mode)； |

该函数的第一个参数是一个普通的路径名，也就是创建后FIFO的名字。第二个参数与打开普通文件的open()函数中的mode 参数相同。 如果mkfifo的第一个参数是一个已经存在的路径名时，会返回EEXIST错误，所以一般典型的调用代码首先会检查是否返回该错误，如果确实返回该错误，那么只要调用打开FIFO的函数就可以了。一般文件的I/O函数都可以用于FIFO，如close、read、write等等。

**2.3有名管道的打开规则**

有名管道比管道多了一个打开操作：open。

FIFO的打开规则：

如果当前打开操作是为读而打开FIFO时，若已经有相应进程为写而打开该FIFO，则当前打开操作将成功返回；否则，可能阻塞直到有相应进程为写而打开该FIFO（当前打开操作设置了阻塞标志）；或者，成功返回（当前打开操作没有设置阻塞标志）。

如果当前打开操作是为写而打开FIFO时，如果已经有相应进程为读而打开该FIFO，则当前打开操作将成功返回；否则，可能阻塞直到有相应进程为读而打开该FIFO（当前打开操作设置了阻塞标志）；或者，返回ENXIO错误（当前打开操作没有设置阻塞标志）。

**2.4有名管道的读写规则**

从FIFO中读取数据：

约定：如果一个进程为了从FIFO中读取数据而阻塞打开FIFO，那么称该进程内的读操作为设置了阻塞标志的读操作。

* 如果有进程写打开FIFO，且当前FIFO内没有数据，则对于设置了阻塞标志的读操作来说，将一直阻塞。对于没有设置阻塞标志读操作来说则返回-1，当前errno值为EAGAIN，提醒以后再试。
* 对于设置了阻塞标志的读操作说，造成阻塞的原因有两种：当前FIFO内有数据，但有其它进程在读这些数据；另外就是FIFO内没有数据。解阻塞的原因则是FIFO中有新的数据写入，不论信写入数据量的大小，也不论读操作请求多少数据量。
* 读打开的阻塞标志只对本进程第一个读操作施加作用，如果本进程内有多个读操作序列，则在第一个读操作被唤醒并完成读操作后，其它将要执行的读操作将不再阻塞，即使在执行读操作时，FIFO中没有数据也一样（此时，读操作返回0）。
* 如果没有进程写打开FIFO，则设置了阻塞标志的读操作会阻塞。

注：如果FIFO中有数据，则设置了阻塞标志的读操作不会因为FIFO中的字节数小于请求读的字节数而阻塞，此时，读操作会返回FIFO中现有的数据量。

向FIFO中写入数据：

约定：如果一个进程为了向FIFO中写入数据而阻塞打开FIFO，那么称该进程内的写操作为设置了阻塞标志的写操作。

对于设置了阻塞标志的写操作：

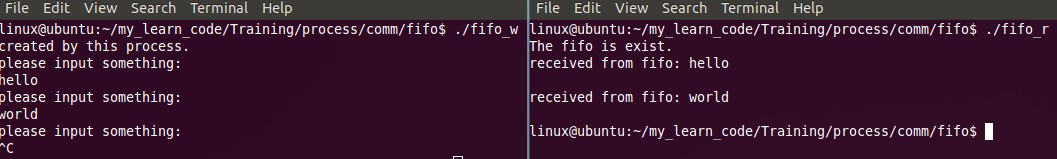
* 当要写入的数据量不大于PIPE\_BUF时，linux将保证写入的原子性。如果此时管道空闲缓冲区不足以容纳要写入的字节数，则进入睡眠，直到当缓冲区中能够容纳要写入的字节数时，才开始进行一次性写操作。
* 当要写入的数据量大于PIPE\_BUF时，linux将不再保证写入的原子性。FIFO缓冲区一有空闲区域，写进程就会试图向管道写入数据，写操作在写完所有请求写的数据后返回。

对于没有设置阻塞标志的写操作：

* 当要写入的数据量大于PIPE\_BUF时，linux将不再保证写入的原子性。在写满所有FIFO空闲缓冲区后，写操作返回。
* 当要写入的数据量不大于PIPE\_BUF时，linux将保证写入的原子性。如果当前FIFO空闲缓冲区能够容纳请求写入的字节数，写完后成功返回；如果当前FIFO空闲缓冲区不能够容纳请求写入的字节数，则返回EAGAIN错误，提醒以后再写；

无代码，无真相

1. /\* name:fifo\_r.c
2. \* 功能：读有名管道，并把读到的数据打印到屏幕\*/
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <unistd.h>
6. #include <sys/types.h>
7. #include <sys/stat.h>
8. #include <errno.h>
9. #include <fcntl.h>
10. #define N 80
11. int main(void) {
12. int in\_file;
13. int count = 1;
14. char buf[N];
15. if((mkfifo("myfifo",0666))<0)//创建有名管道
16. {
17. if(errno==EEXIST)//管道已经存在
18. {
19. printf("The fifo is exist.\n");
20. }
21. else{
22. printf("creat myfifo failed!\n");
23. exit(-1);
24. }
25. }
26. else
27. {
28. printf("created by this process.\n");
29. }
30. in\_file = open("myfifo",O\_RDONLY);
31. if (in\_file < 0) {
32. printf("Error in opening.\n");
33. exit(1);
34. }
35. while ((count = read(in\_file,buf,N)) > 0)
36. {
37. printf("received from fifo: %s\n", buf);
38. memset(buf,0,N);
39. }
40. close(in\_file);
41. return 0;
42. }
43. /\* name:fifo\_w.c
44. \* 功能：从标准输入中读取数据，并到写有名管道\*/
45. #include <stdio.h>
46. #include <stdlib.h>
47. #include <unistd.h>
48. #include <string.h>
49. #include <sys/types.h>
50. #include <sys/stat.h>
51. #include <errno.h>
52. #include <fcntl.h>
53. #define N 80
54. int main() {
55. int out\_file;
56. int nbyte;
57. char buf[N];
58. if((mkfifo("myfifo",0666))<0)    //创建有名管道
59. {
60. if(errno==EEXIST)
61. {
62. printf("The fifo is exist.\n");
63. }
64. else{
65. perror("creat myfifo failed!\n");
66. exit(-1);
67. }
68. }else{
69. printf("created by this process.\n");
70. }
71. out\_file = open("myfifo",O\_WRONLY);
72. if (out\_file < 0) {
73. printf("Error opening fifo.");
74. exit(1);
75. }
76. printf("please input something:\n");
77. while((nbyte = read(0,buf,N))){
78. write(out\_file,buf,nbyte);
79. printf("please input something:\n");
80. }
81. close(out\_file);
82. return 0;
83. }

[](http://blog.chinaunix.net/attachment/201201/18/25365622_1326873029FeDx.jpg)

**小结：**

管道常用于两个方面：（1）在shell中时常会用到管道（作为输入输入的重定向），在这种应用方式下，管道的创建对于用户来说是透明的；（2）用于具有亲缘关系的进程间通信，用户自己创建管道，并完成读写操作。

FIFO可以说是管道的推广，克服了管道无名字的限制，使得无亲缘关系的进程同样可以采用先进先出的通信机制进行通信。

管道和FIFO的数据是字节流，应用程序之间必须事先确定特定的传输"协议"，采用传播具有特定意义的消息。

要灵活应用管道及FIFO，理解它们的读写规则是关键。