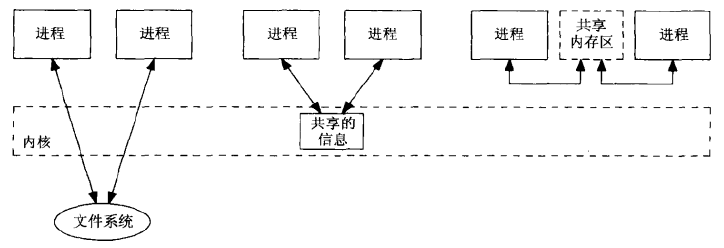
# IPC基础

## IPC原理

每个进程有各自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。所以要么两个进程有默认约定，要么通过内核完成通信。

## 进程信息共享方式



## 匿名管道

匿名管道可用于 IPC（进程间通信），它属于第二种，虽然它使用了文件描述符，但是它没有使用文件系统，数据通过内核缓冲区交换的。其特征是：

  1>、只能进行单向通信。

  2>、只能用于具有血缘关系的进程之间的通信（常用于父子进程之间的通信）。如果没关系，文件描述符不能共享。

  3>、进程的读写操作是以字节流方式来完成。（可以以任意大小读写）

  4>、它是依赖于文件系统的，其生命周期随进程（随进程的结束而结束）。

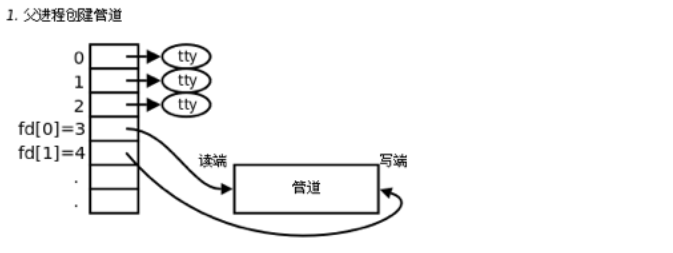
  5>、进程之间有默认的同步关系。

#include<unistd.h>

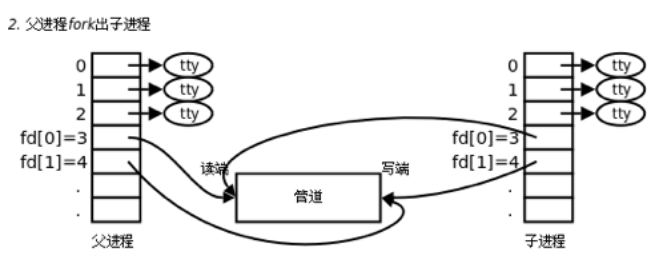
nt pipe(int fd[2]);

其参数：为一个输出型的参数。其中fd[0]中保存读端口的文件描述符，fd[1]中保存写端口的文件描述符，其返回值：成功返回0，失败返回-1。

1. 验证单向通信：（eg:创建一个父子进程，假设子进程用来写数据，父进程用来读数据）（正常的读写，边读边写）

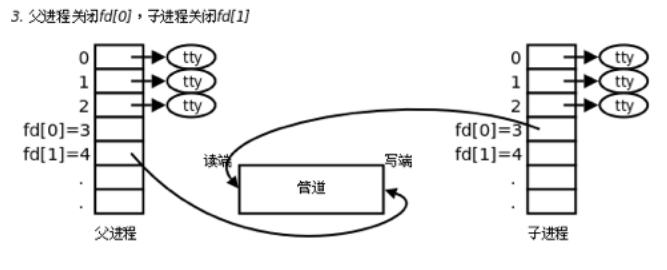


    调用pipe后，fd[0]会指向管道的读端，而fd[1]会指向管道的写端。



    由于子进程会复制父进程的进程地址空间，因此在fork后，会得到上图。

    由于该管道是只能用于单向通信，所以需要把其他不用端口对应的文件描述符关闭。以保证安全性。



代码实现：

2. #include <stdio.h>
3. #include <unistd.h>
4. #include <string.h>
6. **int** main()
7. {
8. **int** fd[2]={0,0};
9. **if**(pipe(fd)==0)
10. {
11. pid\_t id = fork();
12. **if**(id == 0)
13. {//child
14. close(fd[0]);//关闭读
15. **const** **char**\* str="i am a student";
16. **while**(1)
17. {
18. write(fd[1],str,strlen(str));
19. sleep(1); //1s写一次
20. }
21. }
22. **else**
23. {//father
24. close(fd[1]);//关闭写
25. **while**(1)
26. {
27. **char** buf[1024];
28. **int** s = read(fd[0],buf,**sizeof**(buf));
29. **if**(s>0)
30. {
31. buf[s]=0;
32. printf("child  ->  father:%s\n",buf);
33. }
34. }
35. }
36. }
37. **else**
38. {
39. perror("pipe");
40. **return** 1;
41. }
43. **return** 0;
44. }

执行程序后，发现没间隔1s就会输出一次。

2.当父进程不关闭读端口，而每间隔500s读取一次数据时，并且子进程不断的每次以一个字节的数据不停的写：会发现，管道会被写满，子进程写入阻塞，等待父进程读取。

3.当子进程不停的写数据，而父进程读5次数据后，关闭读端口时。子进程会收到SIGPIPE异常终止。

4.当父进程不停的读数据，子进程写5条数据并且关闭写端口后，父进程在读数据的过程中，一旦读完管道中的数据，read就会返回一个0（相当于文件结尾）。并且不再读数据。（如果关闭所以写端口，读端口会一直读，直到文件尾）

linux中的“|”（竖划线）就是一个匿名管道。（没有创建任何文件、并且进程之间具有血缘关系。（都属于bash的子进程））

## 命名管道

与匿名管道的区别：提供了一个路径名与之关联，以FIFO文件的形式存储于文件系统中，能够实现任何两个进程之间通信。而匿名管道对于文件系统是不可见的，它仅限于在父子进程之间的通信。

FIFO是一个设备文件，在文件系统中以文件名的形式存在，因此即使进程与创建FIFO的进程不存在血缘关系也依然可以通信，前提是可以访问该路径。

FIFO(first input first output)总是遵循先进先出的原则，即第一个进来的数据会第一个被读走。

创建命名管道（两种方法）：

#include <sys/stat.h>

int mknod(const char\* path, mode\_t mod, dev\_t dev);

int mkfifo(const char\* path, mode\_t mod);

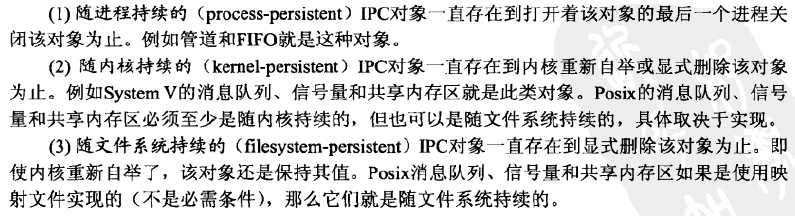
注释：这两个函数都能创建一个FIFO文件，该文件是真实存在于文件系统中的。函数 mknod 中参数 path 为创建命名管道的全路径； mod 为创建命名管道的模式，指的是其存取权限； dev为设备值，改值取决于文件创建的种类，它只在创建设备文件是才会用到。 返回值：这两个函数都是成功返回 0 ，失败返回 -1

命名管道与匿名管道使用的区别：

命名管道创建完成后就可以使用，其使用方法与管道一样，区别在于：命名管道使用之前需要使用open()打开。这是因为：命名管道是设备文件，它是存储在硬盘上的，而管道是存在内存中的特殊文件。但是需要注意的是，命名管道调用open()打开有可能会阻塞，但是如果以读写方式（O\_RDWR）打开则一定不会阻塞；以只读（O\_RDONLY）方式打开时，调用open()的函数会被阻塞直到有数据可读；如果以只写方式（O\_WRONLY）打开时同样也会被阻塞，知道有以读方式打开该管道。

## IPC对象持续性

posix消息队列、信号量、共享内存区可以通过文件系统实现





## IPC对象名字空间



没有名字空间，他就没法做进程间通信，在 linux 系统中，线程间通信也可以看成是进程间通信，这些没有名字的，在 java 中都是线程间通信。