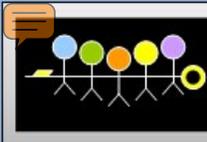


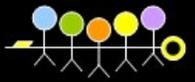


嵌入式系统工程师

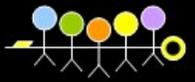




管道、命名管道



- 管道 (pipe)
- 命名管道 (FIFO)

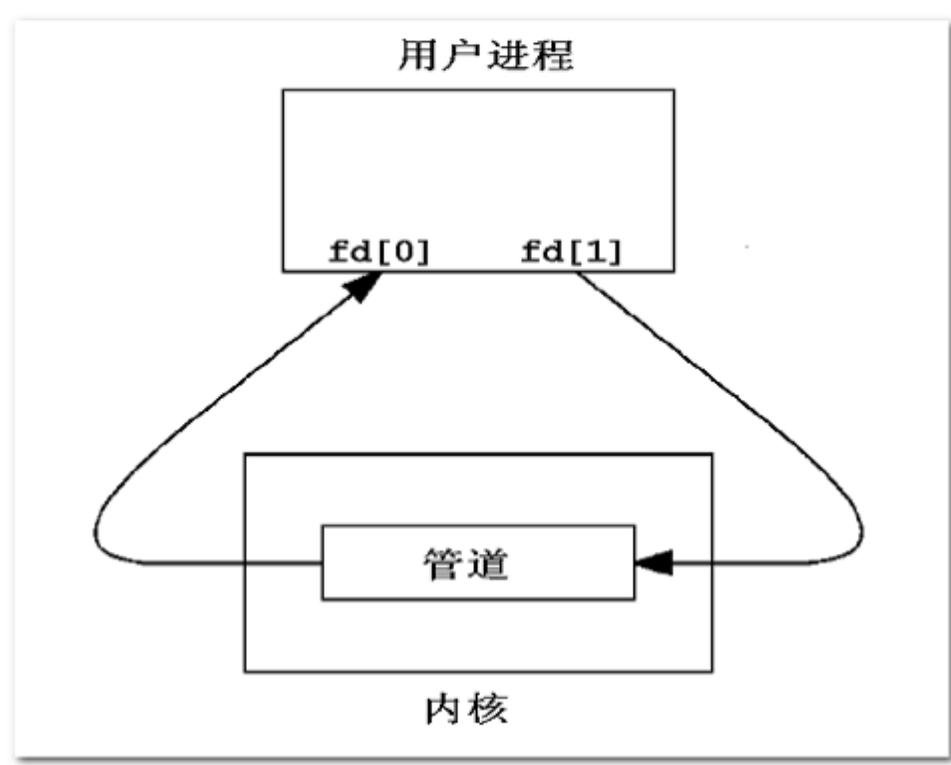


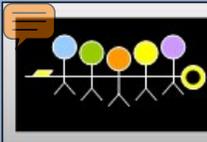
- 管道 (pipe) 概述
- 命名管道 (FIFO)



- 管道 (pipe)
- 命名管道 (FIFO)

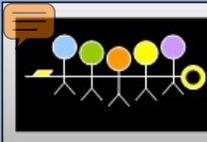
- 管道 (pipe) 又称无名管道。
- 无名管道是一种特殊类型的文件，在应用层体现为两个打开的文件描述符。





管道

- 管道是最古老的UNIX IPC方式，其特点是：
 - 1、半双工，数据在同一时刻只能在一个方向上流动。
 - 2、数据只能从管道的一端写入，从另一端读出。
 - 3、写入管道中的数据遵循先入先出的规则。
 - 4、管道所传送的数据是无格式的，这要求管道的读出方与写入方必须事先约定好数据的格式，如多少字节算一个消息等。



管道

- 5、管道不是普通的文件，不属于某个文件系统，其只存在于内存中。
- 6、管道在内存中对应一个缓冲区。不同的系统其大小不一定相同。
- 7、从管道读数据是一次性操作，数据一旦被读走，它就从管道中被抛弃，释放空间以便写更多的数据。
- 8、管道没有名字，只能在具有公共祖先的进程之间使用。



管道

➤ #include <unistd.h>

```
int pipe(int filedes[2]);
```

功能：经由参数filedes返回两个文件描述符

参数：

➤ filedes为int型数组的首地址，其存放了管道的文件描述符fd[0]、fd[1]。

➤ filedes[0]为读而打开，filedes[1]为写而打开管道，filedes[0]的输出是filedes[1]的输入。

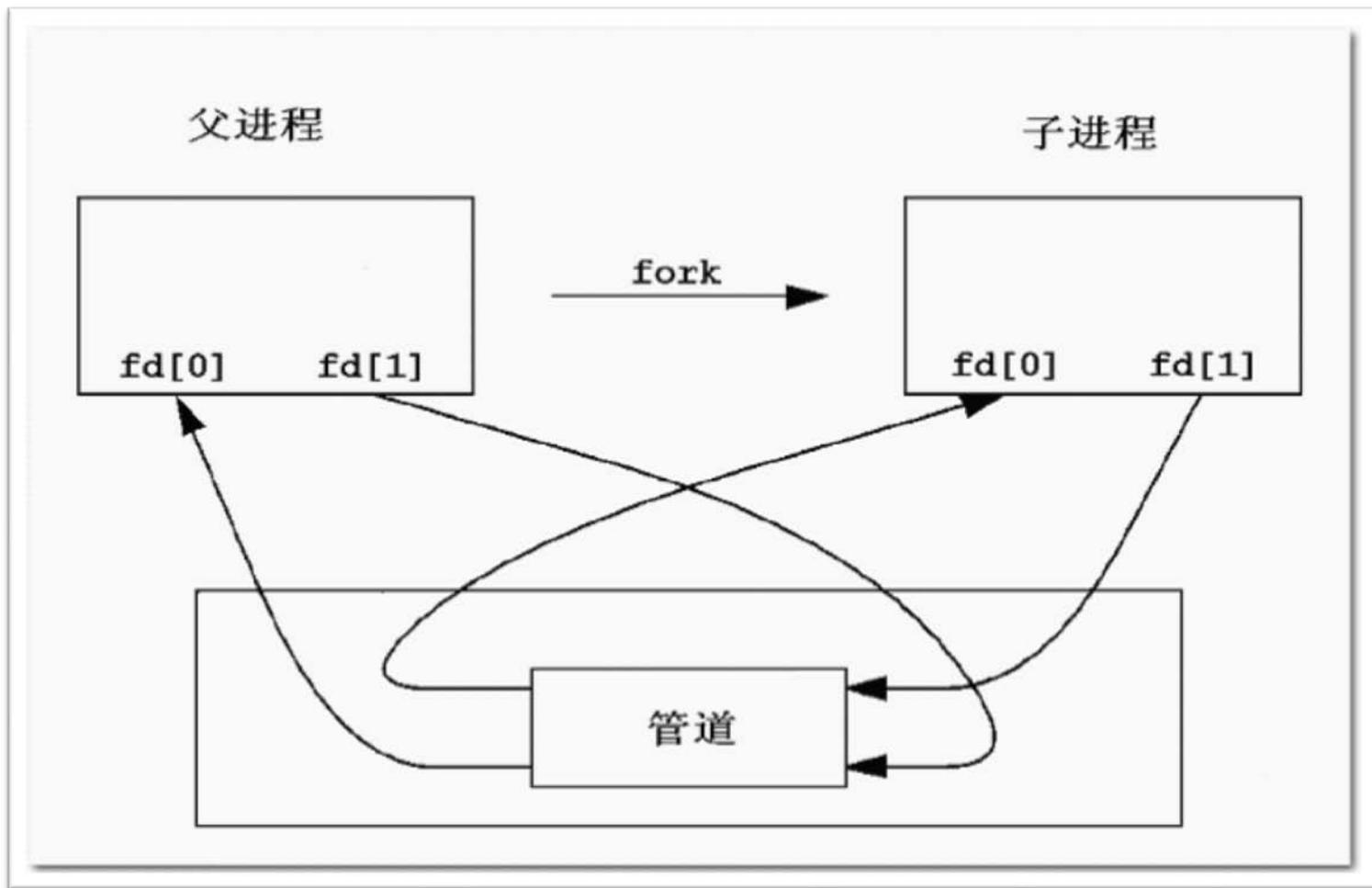
返回值：

成功：返回 0

失败：返回-1

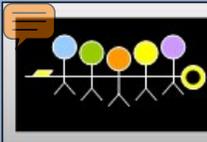
例：[01_pipe_1.c](#)

➤ 父子进程通过管道实现数据的传输



➤ 从管道中读数据的特点

- 1、默认用read函数从管道中读数据是阻塞的。
- 2、调用write函数向管道里写数据，当缓冲区已满时write也会阻塞。
- 3、通信过程中，读端口全部关闭后，写进程向管道内写数据时，写进程会（收到SIGPIPE信号）退出。



➤ 从管道中读数据的特点

编程时可通过 `fcntl` 函数设置文件的阻塞特性。

设置为阻塞：

```
fcntl(fd, F_SETFL, 0);
```

设置为非阻塞：

```
fcntl(fd, F_SETFL, O_NONBLOCK);
```

例：[01_pipe_2.c](#)

➤ 文件描述符概述

- 文件描述符是非负整数，是文件的标识。
- 用户使用文件描述符（file descriptor）来访问文件。
- 利用open打开一个文件时，内核会返回一个文件描述符。

每个进程都有一张文件描述符的表，进程刚被创建时，标准输入、标准输出、标准错误输出设备文件被打开，对应的文件描述符0、1、2记录在表中。

在进程中打开其他文件时，系统会返回文件描述符表中最小可用的文件描述符，并将此文件描述符记录在表中。

➤ 文件描述符概述

注意：

Linux中一个进程最多只能打开NR_OPEN_DEFAULT
(即1024)个文件，故当文件不再使用时应及时调用
close函数关闭文件。

➤ 文件描述符的复制

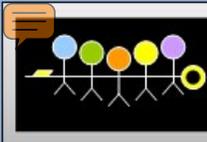
dup和dup2是两个非常有用的系统调用，都是用来复制一个文件的描述符，使新的文件描述符也标识旧的文件描述符所标识的文件。

➤ `int dup(int oldfd);`

➤ `int dup2(int oldfd, int newfd);`

➤ dup和dup2经常用来重定向进程的stdin、stdout和stderr。

回顾: `ls > test.txt`



➤ dup函数

```
#include <unistd.h>  
int dup(int oldfd);
```

功能:

复制oldfd文件描述符，并分配一个新的文件描述符，新的文件描述符是调用进程文件描述符表中最小可用的文件描述符。

参数：要复制的文件描述符oldfd。

返回值:

成功：新文件描述符。

失败：返回-1，错误代码存于errno中。

例：[02_dup.c](#)

➤ dup2函数

```
#include <unistd.h>
```

```
int dup2(int oldfd, int newfd)
```

功能:

复制一份打开的文件描述符oldfd，并分配新的文件描述符newfd，newfd也标识oldfd所标识的文件。

注意:

newfd是小于文件描述符最大允许值的非负整数，如果newfd是一个已经打开的文件描述符，则首先关闭该文件，然后再复制。

➤ dup2函数

参数:

- 要复制的文件描述符oldfd
- 分配的新的文件描述符newfd

返回值:

- 成功: 返回newfd
- 失败: 返回-1, 错误代码存于errno中

例: 03_dup2.c

➤ 复制文件描述符后新旧文件描述符的特点

使用dup或dup2复制文件描述符后，新文件描述符和旧文件描述符指向同一个文件，共享文件锁定、读写位置和各项权限。

当关闭新的文件描述符时，通过旧文件描述符仍可操作文件。

当关闭旧的文件描述符时，通过新的文件描述符仍可操作文件。

➤ exec前后文件描述符的特点

`close_on_exec`标志决定了文件描述符在执行exec后文件描述符是否可用。

文件描述符的`close_on_exec`标志默认是关闭的，即文件描述符在执行exec后文件描述符是可用的。

若没有设置`close_on_exec`标志位，进程中打开的文件描述符，及其相关的设置在exec后不变，可供新启动的程序使用。

➤ exec前后文件描述符的特点

设置close_on_exec标志位的方法:

```
int flags;
```

```
flags = fcntl(fd, F_GETFD); // 获得标志
```

```
flags |= FD_CLOEXEC; // 打开标志位
```

```
flags &= ~FD_CLOEXEC; // 关闭标志位
```

```
fcntl(fd, F_SETFD, flags); // 设置标志
```

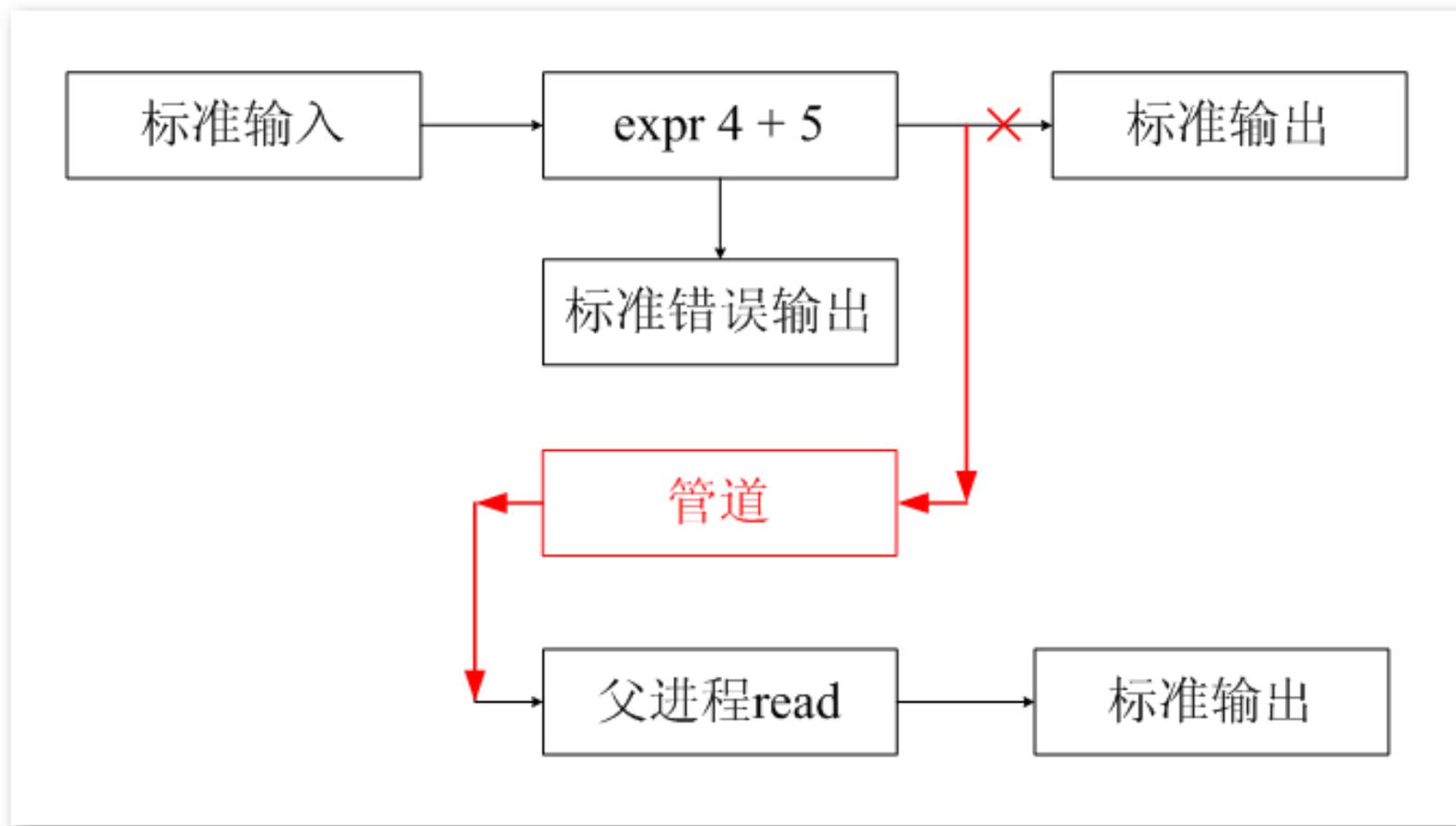
➤ 练习

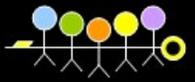
题目: 借用外部命令, 实现计算器功能

➤ 提示:

- `expr` 是个外部命令, 它向标准输出打印运算结果。
- 创建一个管道以便让 `expr 4 + 5` 的输出到管道中
- 子进程 `exec` 执行 `expr 4 + 5` 命令之前重定向“标准输出”到“管道写端”。
- 父进程从管道读端读取数据, 并显示运算结果

外部命令结果信息输出至管道:





- 管道 (pipe)
- 命名管道 (FIFO)

命名管道 (FIFO)

- 命名管道 (FIFO) 和管道 (pipe) 基本相同，但也有一些显著的不同，其特点是：
 - 1、半双工，数据在同一时刻只能在一个方向上流动。
 - 2、写入FIFO中的数据遵循先入先出的规则。
 - 3、FIFO所传送的数据是无格式的，这要求FIFO的读出方与写入方必须事先约定好数据的格式，如多少字节算一个消息等。
 - 4、FIFO在文件系统中作为一个特殊的文件而存在，但FIFO中的内容却存放在内存中。

命名管道 (FIFO)

- 5、管道在内存中对应一个缓冲区。不同的系统其大小不一定相同。
- 6、从FIFO读数据是一次性操作，数据一旦被读，它就从FIFO中被抛弃，释放空间以便写更多的数据。
- 7、当使用FIFO的进程退出后，FIFO文件将继续保存在文件系统中以便以后使用。
- 8、FIFO有名字，不相关的进程可以通过打开命名管道进行通信。

命名管道 (FIFO)

➤ FIFO文件的创建

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mkfifo( const char *pathname, mode_t mode);
```

参数:

pathname: FIFO的路径名+文件名。

mode: mode_t类型的权限描述符。

返回值:

成功: 返回 0

失败: 如果文件已经存在, 则会出错且返回-1。

命名管道 (FIFO)

➤ 操作FIFO文件时的特点

系统调用的I/O函数都可以作用于FIFO，如open、close、read、write等。

➤ 打开FIFO时，非阻塞标志(O_NONBLOCK)产生下列影响：

➤ 特点一：

➤ 不指定O_NONBLOCK (即open没有位或O_NONBLOCK)

1、open以只读方式打开FIFO时，要阻塞到某个进程为写而打开此FIFO

2、open以只写方式打开FIFO时，要阻塞到某个进程为读而打开此FIFO。

例：04_fifo_read_1.c 04_fifo_write_1.c

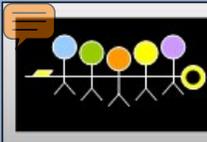
命名管道 (FIFO)

3、open以只读、只写方式打开FIFO时会阻塞，调用read函数从FIFO里读数据时read也会阻塞。

例: 04_fifo_read_2.c 04_fifo_write_2.c

4、通信过程中若写进程先退出了，则调用read函数从FIFO里读数据时不阻塞；若写进程又重新运行，则调用read函数从FIFO里读数据时又恢复阻塞。

例: 04_fifo_read_3.c 04_fifo_write_3.c



命名管道 (FIFO)

5、通信过程中，读进程退出后，写进程向命名管道内写数据时，写进程也会（收到SIGPIPE信号）退出。

例：04_fifo_read_4.c 04_fifo_write_4.c

6、调用write函数向FIFO里写数据，当缓冲区已满时write也会阻塞。

命名管道 (FIFO)

- 打开FIFO时，非阻塞标志 (O_NONBLOCK) 产生下列影响：
 - 特点二：
 - 指定O_NONBLOCK (即open位或O_NONBLOCK)
 - 1、先以只读方式打开：如果没有进程已经为写而打开一个FIFO，只读open成功，并且open不阻塞。
 - 2、先以只写方式打开：如果没有进程已经为读而打开一个FIFO，只写open将出错返回-1。
 - 3、read、write读写命名管道中读数据时不阻塞。
 - 4、通信过程中，读进程退出后，写进程向命名管道内写数据时，写进程也会（收到SIGPIPE信号）退出。
- 例：04_fifo_read_5.c 04_fifo_write_5.c

命名管道 (FIFO)

► 注意:

open函数以可读可写方式打开FIFO文件时的特点:

- 1、open不阻塞。
- 2、调用read函数从FIFO里读数据时read会阻塞。
- 3、调用write函数向FIFO里写数据，当缓冲区已满时write也会阻塞。

命名管道 (FIFO)

➤ 练习

➤ 题目：实现单机QQ聊天

➤ 提示：

➤ 父进程创建子进程，实现多任务。

父进程负责发信息(向FIFO里写数据)，子进程负责接收信息(从FIFO里读数据)。

➤ 打开命名管道的用阻塞的方法打开。

命名管道 (FIFO)

➤ QQ聊天框架及流程图

