

嵌入式系统工程师

# 管道、命名管道

- 进程间通信概述
- 管道(pipe)
- 命名管道(FIFO)

- 进程间通信概述
- 管道(pipe)
- 命名管道(FIFO)

# 进程间通信概述

## ➤ 进程间通信 (IPC: Inter Processes Communication)

进程是一个独立的资源分配单元，不同进程（这里所说的进程通常指的是用户进程）之间的资源是独立的，没有关联，不能在一个进程中直接访问另一个进程的资源（例如打开的文件描述符）。

进程不是孤立的，不同的进程需要进行信息的交互和状态的传递等，因此需要进程间通信。

# 进程间通信概述

## ➤ 进程间通信功能:

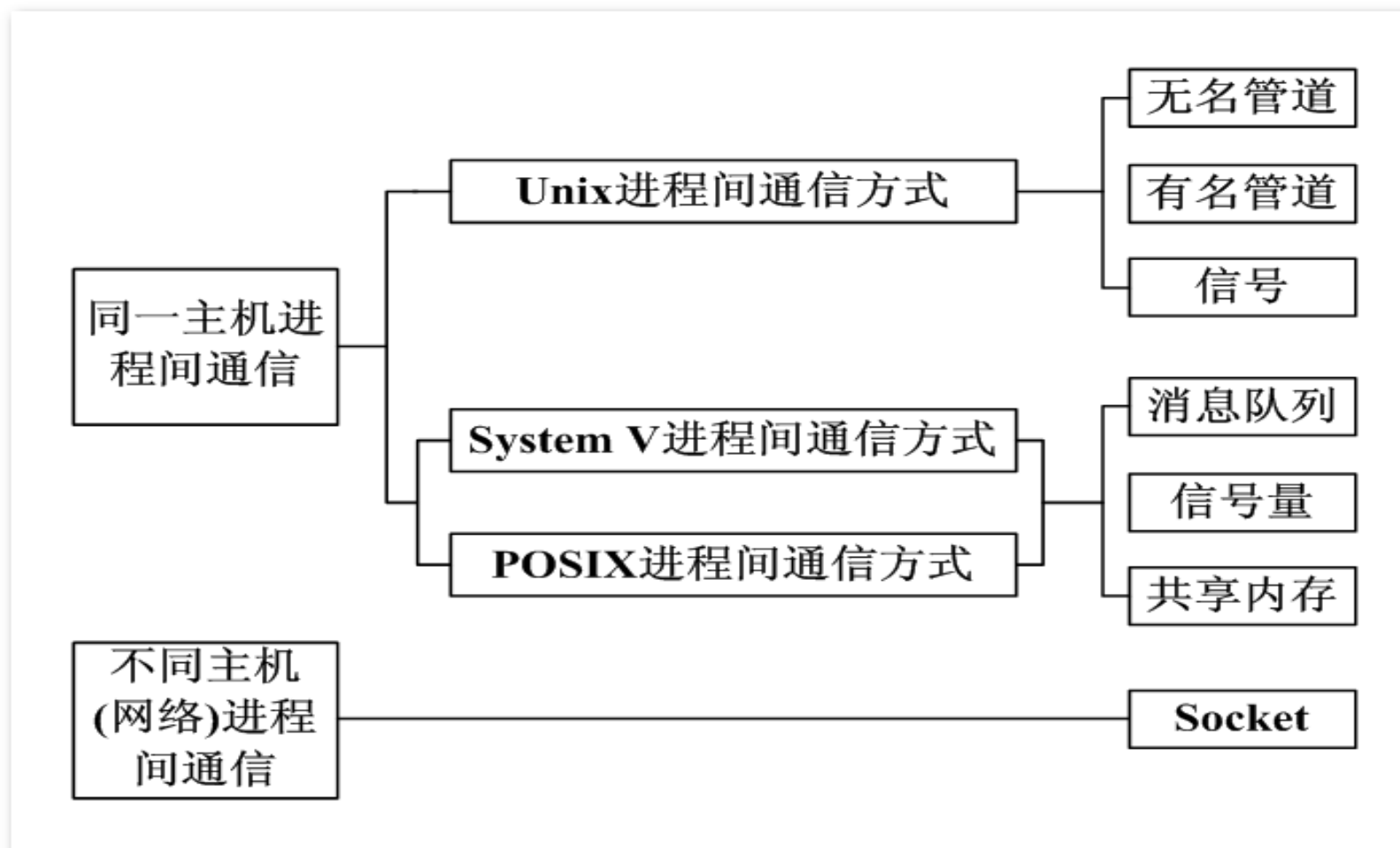
- 数据传输: 一个进程需要将它的数据发送给另一个进程。
- 资源共享: 多个进程之间共享同样的资源。
- 通知事件: 一个进程需要向另一个或一组进程发送消息, 通知它们发生了某种事件。
- 进程控制: 有些进程希望完全控制另一个进程的执行 (如Debug进程), 此时控制进程希望能够拦截另一个进程的所有操作, 并能够及时知道它的状态改变。

# 进程间通信概述

- linux进程间通信 (IPC) 由以下几个部分发展而来
  - 最初的UNIX进程间通信
  - SYSTEM V进程间通信
  - POSIX进程间通信 (POSIX: Portable Operating System interface可移植操作系统接口 )
  - Socket进程间通信
- Linux把优势都继承了下来并形成了自己的IPC

# 进程间通信概述

## ➤ Linux操作系统支持的主要进程间通信的通信机制

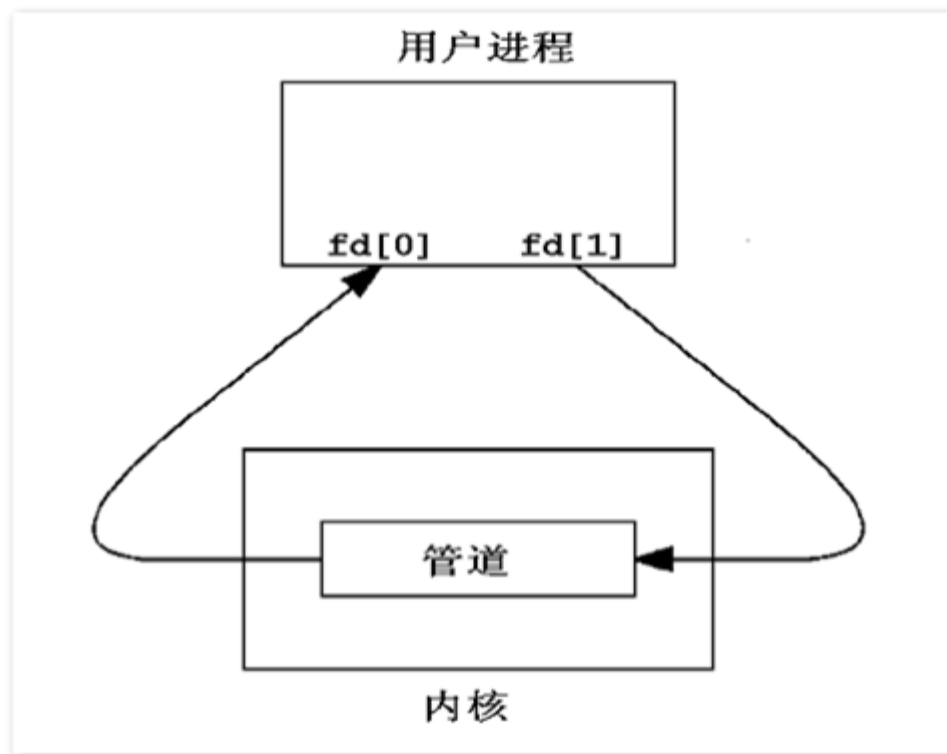




- 进程间通信概述
- 管道(pipe)
- 命名管道(FIFO)

# 管道

- 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：
  - 半双工，数据在同一时刻只能在一个方向上流动。
  - 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。
  - 管道没有名字，只能在具有公共祖先的进程之间使用。
  - 管道的缓冲区是有限的。管道是一个固定大小的缓冲区。在Linux中，该缓冲区的大小为4Kbyte。

# 管道

- 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。
- 数据只能从管道的一端写入，从另一端读出。
- 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。

➤ `#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。

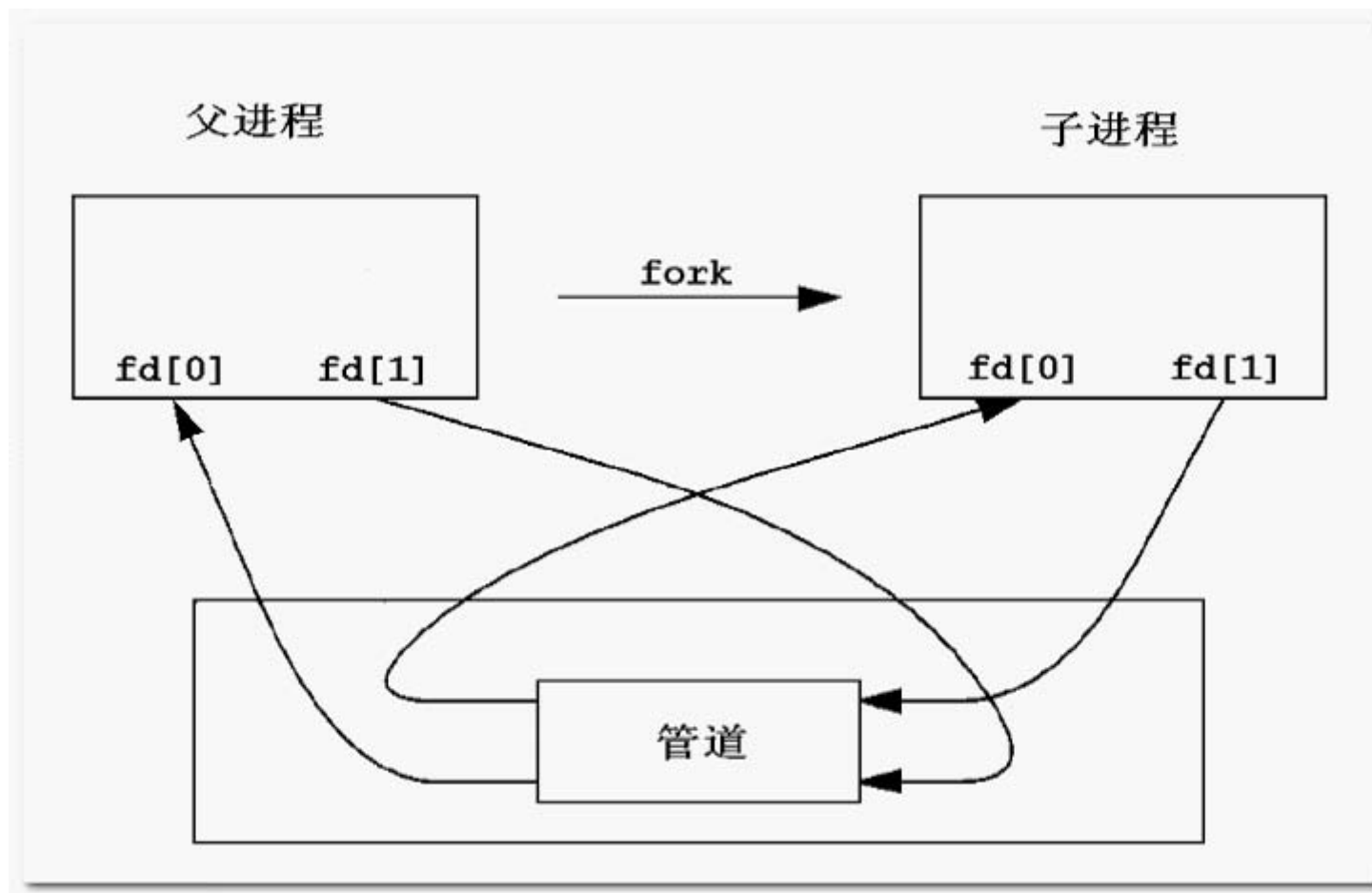
返回值：

成功：返回 0

失败：返回-1

# 管道

- 父子进程通过管道实现数据的传输



## ➤ 文件描述符概述

- 文件描述符是非负整数，是文件的标识。
- 内核利用文件描述符（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，其计录了默认打开的标准输入、标准输出、标准错误输出三个设备文件的文件描述符0、1、2。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。

- 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。
  - `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。



## ➤ 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:

oldfd: 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。

➤ dup函数

➤ 注意:

新的文件描述符和oldfd指向的是同一个文件，共享oldfd所有的锁定、读写位置和各项权限或标志，但文件描述符之间并不共享close-on-exec标志。

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:

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](#)

- 进程间通信概述
- 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：
  - FIFO 在文件系统中作为一个特殊的文件而存在。
  - 虽然 FIFO 文件存在于文件系统中，但 FIFO 中的内容却存放在内存中，在 Linux 中，该缓冲区的大小为 4Kbyte。
  - FIFO 有名字，不同的进程可以通过该命名管道进行通信。

# 命名管道(FIFO)

- FIFO所传送的数据是无格式的。
- 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。
- 当共享FIFO的进程执行完所有的I/O操作以后，FIFO将继续保存在文件系统中以便以后使用。

# 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)

## ➤ FIFO文件的读写

- 因为使用pipe的进程通过继承获得了pipe的文件描述符，所以pipe仅需要创建而不需要打开。
- 但是FIFO则需要打开，因为使用它们的进程可以没有任何关系。
- 一般文件的I/O函数都可以作用于FIFO，如open、close、read、write等。

# 命名管道(FIFO)

## ➤ FIFO文件的读写

- 当打开FIFO时，非阻塞标志(O\_NONBLOCK)产生下列影响：
- 不指定O\_NONBLOCK (即不写 | O\_NONBLOCK)：
  - 只读open要阻塞到某个其他进程为写而打开此FIFO。
  - 只写open要阻塞到某个其他进程为读而打开此FIFO。

例: 04\_fifo\_read\_1.c    04\_fifo\_write\_1.c

# 命名管道(FIFO)

## ➤ FIFO文件的读写

## ➤ 注:

不指定O\_NONBLOCK时，除了只读、只写open会阻塞，调用read函数从FIFO里读数据时read也会阻塞。

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。

通信过程中，读进程退出后，写进程向命名管道内写数据时，写进程也会退出。


例: [04\\_fifo\\_read\\_2.c](#)    [04\\_fifo\\_write\\_2.c](#)

# 命名管道(FIFO)

## ➤ FIFO文件的读写

### 先运行读命名管道程序

```
[root@localhost pipe_fifo]# ./read
read from my_fifo buf=[Hello I love you]
read from my_fifo buf=[]
read from my_fifo buf=[]
read from my_fifo buf=[]
read from my_fifo buf=[Hello I love you]
|
```



### 再运行写命名管道程序

```
[root@localhost pipe_fifo]# ./write
write to my_fifo buf=Hello I love you
```

### 重新运行写程序

```
[root@localhost pipe_fifo]# ./write
write to my_fifo buf=Hello I love you
|
```

# 命名管道(FIFO)

- FIFO文件的读写

- 指定O\_NONBLOCK:

- 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, 立即返回。

- 只写方式打开: 如果没有进程已经为读而打开一个FIFO, 那么只写open将出错返回-1。

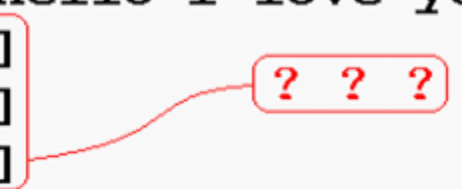
- 例: [04\\_fifo\\_read\\_3.c](#) [04\\_fifo\\_write\\_3.c](#)

# 命名管道(FIFO)

## ➤ FIFO文件的读写

### 先运行读命名管道程序

```
[root@localhost pipe_fifo]# ./read
read from my_fifo buf=[Hello I love you]
read from my_fifo buf=[]
read from my_fifo buf=[]
read from my_fifo buf=[]
read from my_fifo buf=[Hello I love you]
|
```



### 再运行写命名管道程序

```
[root@localhost pipe_fifo]# ./write
write to my_fifo buf=Hello I love you
```

### 重新运行写程序

```
[root@localhost pipe_fifo]# ./write
write to my_fifo buf=Hello I love you
|
```

