

实验3、基于管道的进程间数据传输

实验目的

- ①熟悉Linux下的应用程序开发
- ②熟悉Linux的进程控制原语的使用
- ③掌握Linux操作系统的进程间通信机制管道的使用。
- ④掌握Linux操作系统中父进程与子进程的同步。

实验内容

- 父进程首先使用系统调用`pipe()`建立一个管道，然后使用系统调用`fork()` 创建子进程1
- 子进程1关闭管道读文件
- 子进程1通过文件I/O操作向管道写文件写一句话（向文件中写入字符串）：

Child process 1 is sending a message!

- 然后子进程1调用`exit ()` 结束运行

实验内容

- 父进程再次使用系统调用`fork()` 创建子进程2
- 子进程2关闭管道读文件
- 子进程2通过文件I/O操作向管道写文件写一句话（向文件中写入字符串）：

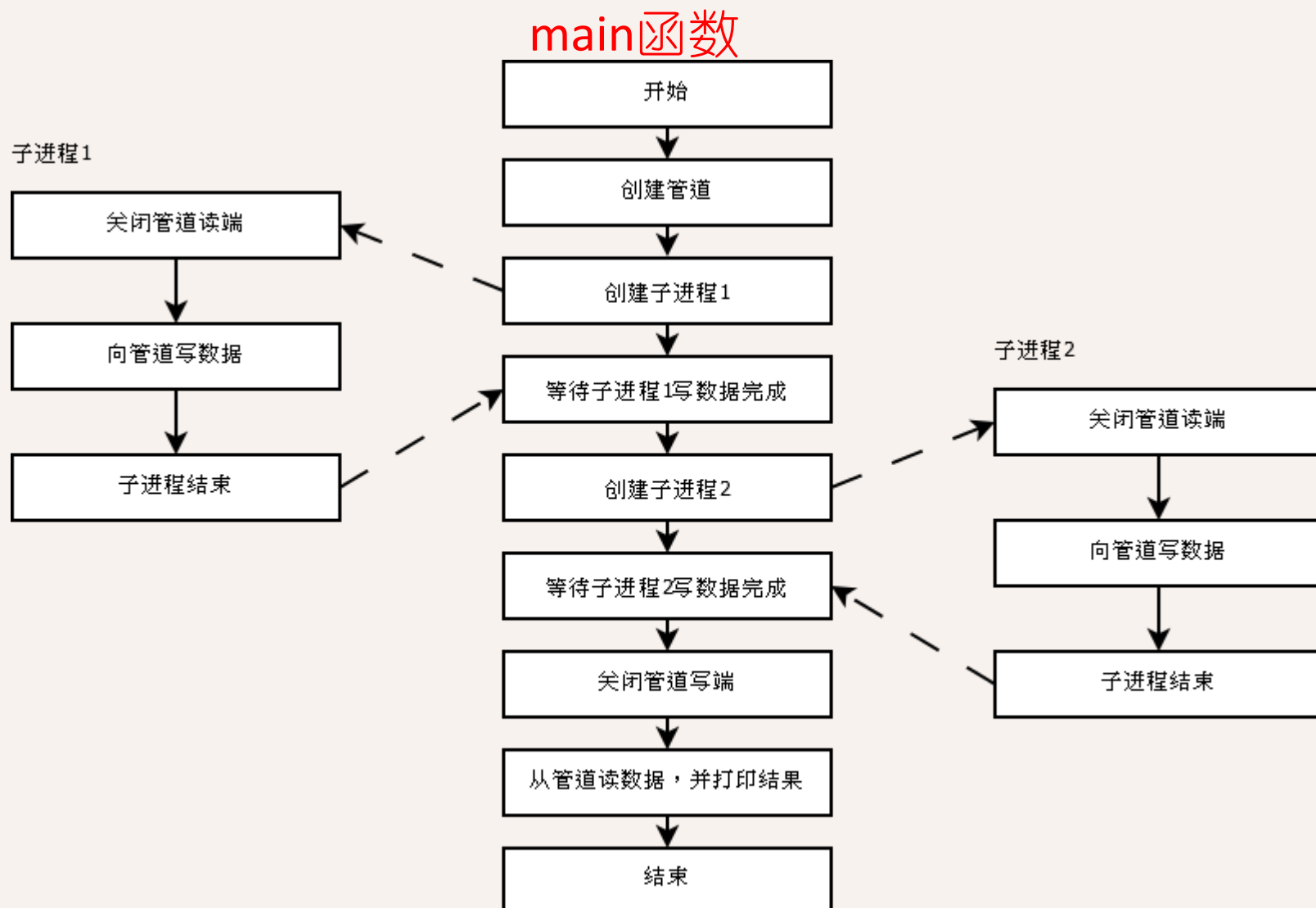
Child process 2 is sending a message!

- 然后子进程2调用`exit ()` 结束运行

实验内容

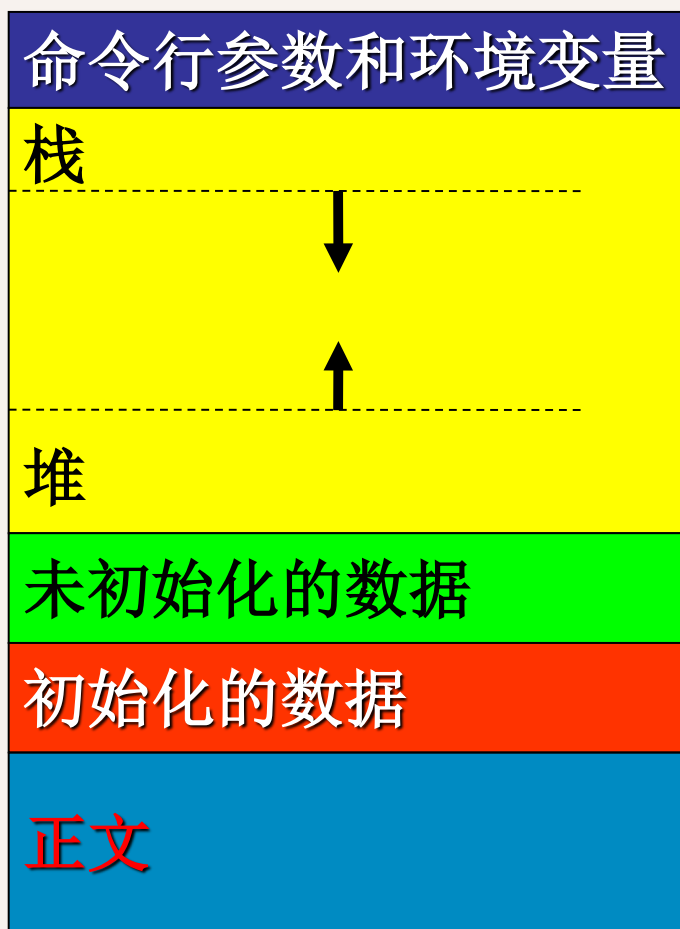
- 父进程关闭管道写文件
- 父进程通过文件I/O操作从管道读文件中读出来自于两个子进程的信息，通过printf语句打印输出在屏幕上

● 实验原理



Linux进程的内存空间布局

High address



创建进程函数

- 函数原型

- `pid_t fork(void); (unistd.h)`

- 返回值

- **fork函数被正确调用后，将会返回两次！！**（通过返回值，可以确定是在父进程中返回还是子进程中返回）
 - 在子进程中返回其返回值为0（不合法的PID，因为PID=0的进程是交换进程）
 - 在父进程中返回其返回值为子进程ID（可以让父进程知道所创建的子进程ID号）
 - 出错返回-1

示例代码：fork

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

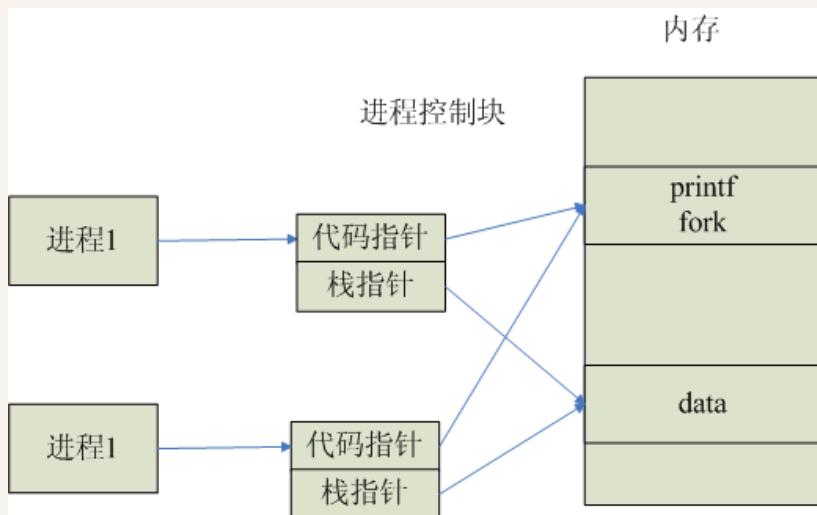
int main(void)
{
    pid_t pid;
    pid=fork();
    if(pid==-1)
        printf(“fork error\n”);
    else if(pid==0)
        printf(“in the child process\n”);
    else
        print(“in the parent process, the child process id is
            %d\n”,pid);
    return 0;}
```


父子进程的数据共享

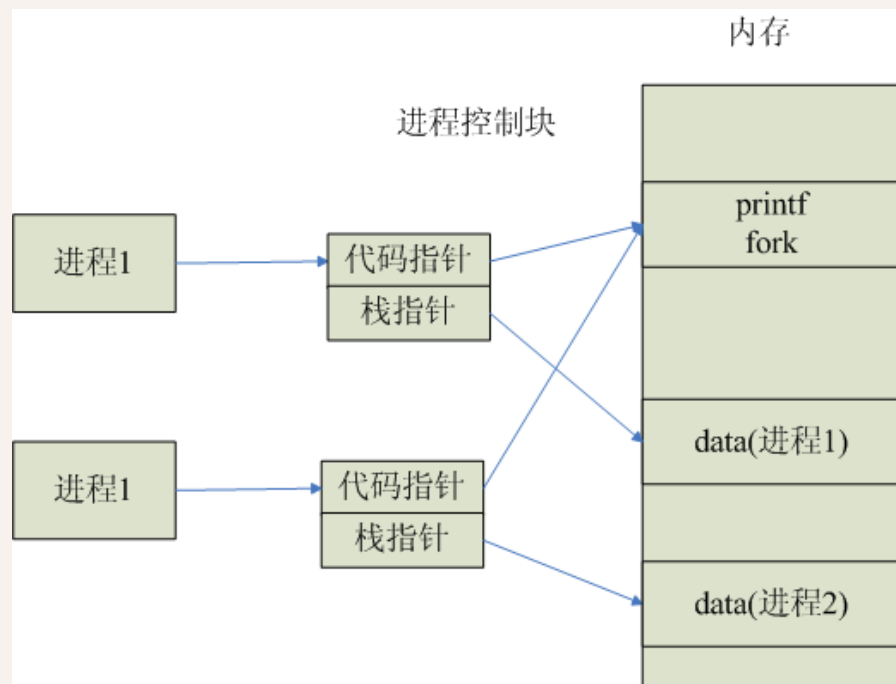
- 子进程和父进程继续执行fork调用之后的指令
- 子进程是父进程的副本
 - 子进程获得父进程数据段、堆和栈的副本（拷贝）
 - 父子进程共享正文段（只读的）
- 为了提高效率，fork后并不立即复制父进程，采用了写时复制机制（Copy-On-Write）
 - 当父子进程任意之一要修改数据段、堆、栈时，进行复制操作，并且仅复制修改区域

父子进程的数据共享

1. 父子进程均无写操作时



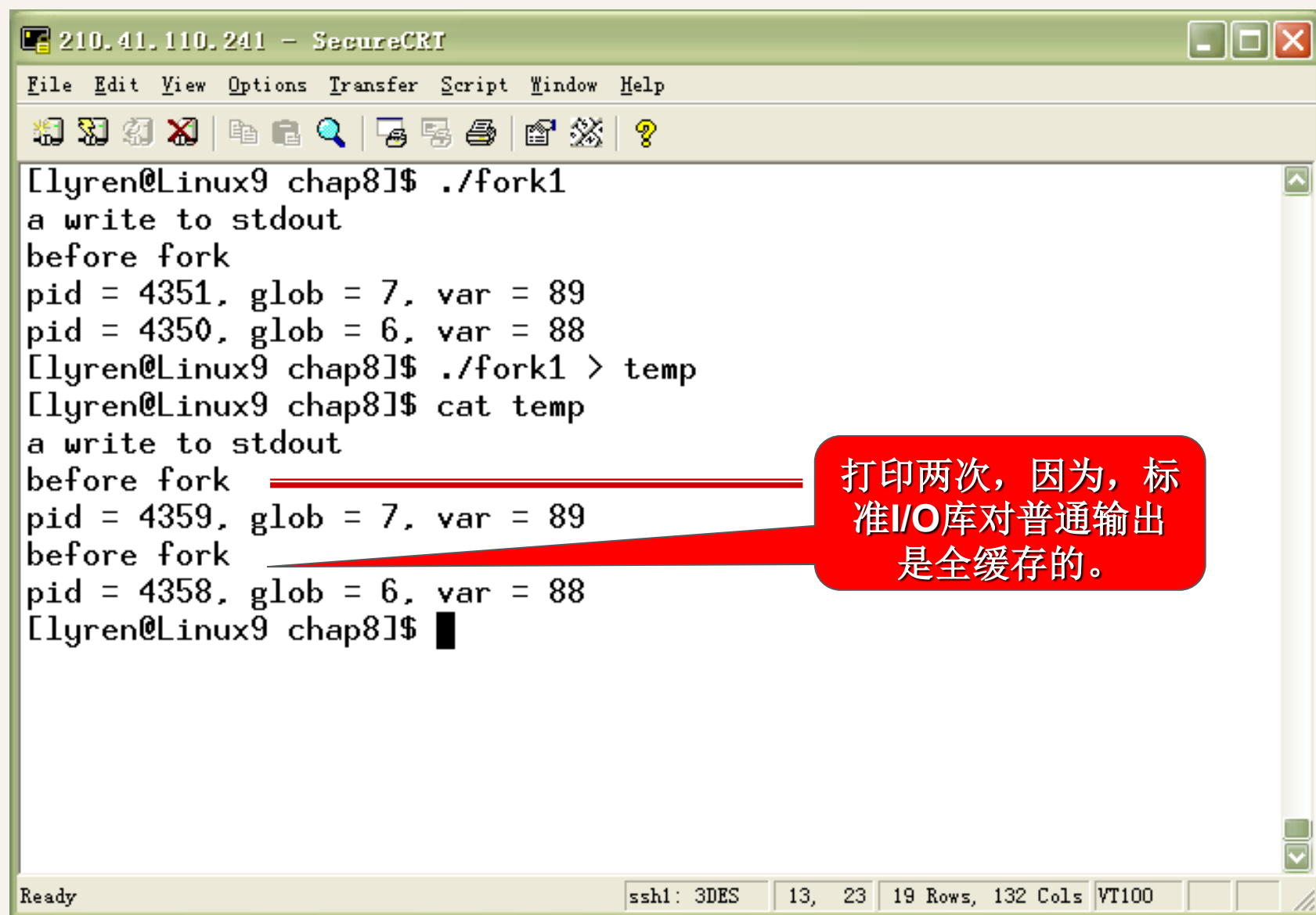
2. 父进程或者子进程对数据有修改操作时，子进程或父进程将新产生一份进程的数据拷贝，然后再修改。



创建子进程及父子进程共享数据示例

```
int    glob = 6;           全局变量，在数据段中
char   buf[ ] = "a write to stdout\n";
int    main(void)
{
    int    var;             局部变量，在栈中
    pid_t  pid;
    var = 88;
    if (write(STDOUT_FILENO, buf, sizeof(buf)-1) != sizeof(buf)-1)
        err_sys("write error");
    printf("before fork\n");
    if ( (pid = fork()) < 0) err_sys("fork error");
    else if (pid == 0)
        glob++; 在子进程中修改全局变量和局部变量并输出变量值
        var++;
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);}
    else{
        sleep(2); 在父进程中输出变量值
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);}
    exit(0);}
```

创建子进程及父子进程共享数据示例



```
210.41.110.241 - SecureCRT
File Edit View Options Transfer Script Window Help
a write to stdout
before fork
pid = 4351, glob = 7, var = 89
pid = 4350, glob = 6, var = 88
[lyren@Linux9 chap8]$ ./fork1 > temp
[lyren@Linux9 chap8]$ cat temp
a write to stdout
before fork
pid = 4359, glob = 7, var = 89
before fork
pid = 4358, glob = 6, var = 88
[lyren@Linux9 chap8]$
```

打印两次，因为，标准I/O库对普通输出是全缓存的。

Ready ssh1: 3DES 13, 23 19 Rows, 132 Cols VT100

特殊文件-管道

- 最古老、最简单的UNIX进程间通信机制
- 管道是一种特殊文件
- 管道的局限性
 - 半双工，只能一个进程写，一个进程读
 - 只能在父子进程之间使用

管道的创建

- `pipe`函数创建一个通信缓冲区，程序可以通过文件描述符`filides[0]`和`filides[1]`来访问这个缓冲区
- `filides[0]`为读而打开，`filides[1]`为写而打开
- 写入`filides[1]`的数据可以按照先进先出的顺序从`filides[0]`中读出

```
#include <unistd.h>
int pipe(int filides[2]);
```

■ 如果成功则返回0。否则返回-1，并设置`errno`。

管道的使用方法

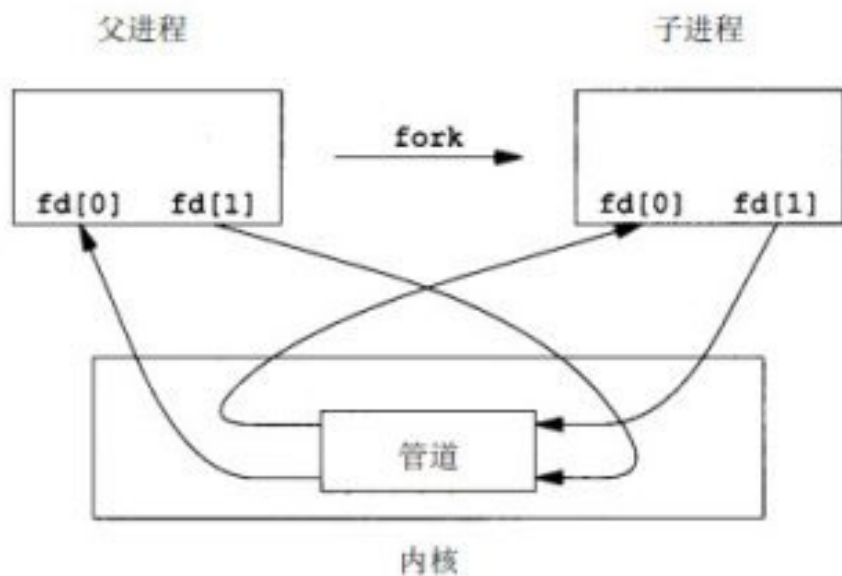


图14-2 fork之后的半双工管道

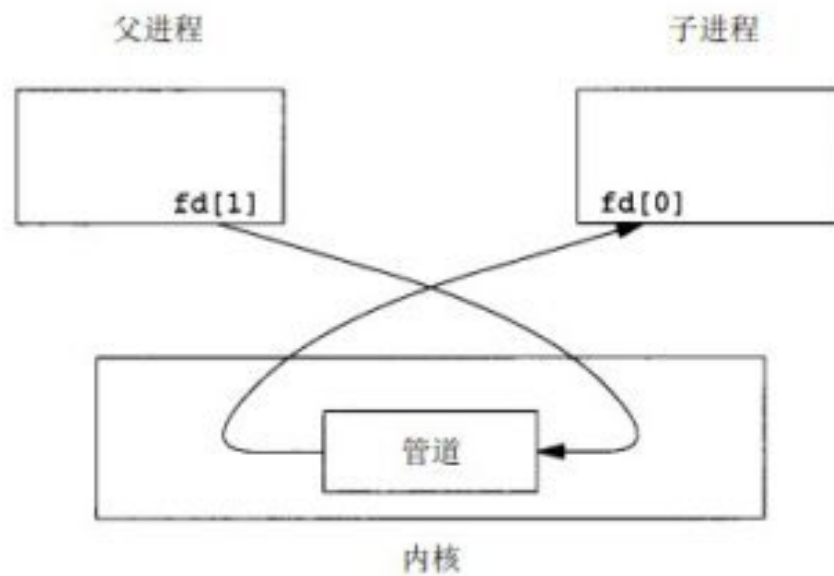


图14-3 从父进程到子进程的管道

读管道-read函数

头文件unistd.h

```
ssize_t read( int fildes, void *buf, size_t nbytes);
```

- 第一个参数为管道读文件描述符（pipe函数获取的fildes[0]）
- 第二个参数为读缓冲区的指针，第三个参数为希望读取的字节数
- read函数从打开的管道文件中读数据。如成功，则返回实际读到的字节数，如已到达管道文件的末尾或无数据可读，则返回0；

写管道-write函数

头文件unistd.h

ssize_t write(int fildes, const void *buf, size_t nbytes);

- 第一个参数为管道写文件描述符（pipe函数获取的fildes[1]）
- 第二个参数为写缓冲区的指针，第三个参数为希望写入的字节数
- 该函数返回实际写的字节数，通常与参数nbytes的值相同，否则表示出错。

关闭管道-close函数

头文件unistd.h

```
int close( int fildes );
```

- 该函数关闭管道的读文件和写文件，参数为读文件和写文件的文件描述符
- 进程关闭管道读/写文件后，就不能再通过该文件描述符读/写管道

管道程序示例

- 程序中父进程向管道写入一个字符串，子进程读出这个字符串

```
int main(void) {  
    char bufin[BUFSIZE] = "empty";  
    char bufout[] = "hello";  
    int bytesin;  
    pid_t childpid;  
    int fd[2];  
    if (pipe(fd) == -1) {  
        perror("Failed to create the pipe");  
        return 1;}  
}
```

管道程序示例

```
bytesin = strlen(bufin);
childpid = fork();
if (childpid == -1) {
    perror("Failed to fork");
    return 1;}
if (childpid) /* parent code */
    write(fd[1], bufout, strlen(bufout)+1);
else /* child code */
    bytesin = read(fd[0], bufin, BUFSIZE);
fprintf(stderr, "[%ld]:my bufin is {%.*s}, my bufout is
    {%s}\n", (long)getpid(), bytesin, bufin, bufout);
return 0;
}
```

- 实验报告

说明实验过程。

进行结果分析。