Lab4-Linux进程控制_pipe()

学号	姓名	专业班级	课程名称	学期	任课教师	完成日期
23020011046	翟一航	软件工程23	操作系统	2025秋	李晓慧	1.10

一、实验目的及要求

- 了解进程与程序的区别，加深对进程概念的理解；
- 学习如何利用管道机制、消息缓冲队列、共享存储区机制进行进程间的通信，并加深对上述通信机制的理解；
- 学习 `pipe()` 函数使用原理和范例,以实现进程通信。

二、实验内容与源码

`pipe()` 头文件及调用

```
# include <unistd.h>

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

`write()` 与 `read()`

```
int write(int pipe_fd, char *buffer, size_t len);
int read(int pipe_fd, char *buffer, size_t len);
```

函数说明

- `pipe()` ：建立管道
  - 建立管道，并将文件描述符由数组filedes 返回。
  - filedes[0]：管道的读取端，用read从管道中读数据
  - filedes[1]：管道的写入端，用write向管道中写数据。
  - 返回值：若成功则返回0，否则返回-1
- 基本步骤
  - 利用 `pipe` 创建管道；
  - 利用 `fork` 创建子进程
  - 子进程利用 `write` 向管道写入数据
    - `char *str=“it is sending a message to parent!”;`
    - `close(filedes[0]); //关闭读端`
    - `write(filedes[1], str, strlen(str)); //写数据`
  - 父进程利用 `read` 从管道读取数据
    - `close(filedes[1]); //关闭写端`

        - `read(filedes[0], buf, sizeof(buf)); //读数据`
    - 父进程显示接收到的数据，及数据的长度
- `pipe()` 使用示例

```c
# include <unistd.h>
# include <stdio.h>
# include <sys/wait.h>

int main( void )
{
    int fd[2];
    char buf[80];
    pid_t pid;

    pipe(fd);

    if ((pid=fork()) > 0)
    {
        printf( "This is in the father process,here write a string to the pipe.\n" );
        char s[] = "Hello world , this is write by pipe.\n";
        write( fd[1], s, sizeof(s) );
        close( fd[0] );
        close( fd[1] );
    }
    else
    {
        printf( "This is in the child process,here read a string from the pipe.\n" );
        read(fd[0], buf, sizeof(buf));
        printf( "%s\n", buf );
        close(fd[0]);
        close(fd[1]);
    }

    waitpid(pid, NULL, 0);
    return 0;
}
```

Task—练习1

- 运行以上示例程序，观察并分析运行结果

Task—练习2

- 编写一C语言程序，使其用管道来实现父子进程间通信。子进程向父进程发送字符串“it is sending a message to parent!”；父进程则从管道中读出子进程发来的消息，并将其显示到屏幕上，然后终止。

- 运行该程序，观察、记录并简单分析其运行结果。

三、实验结果分析

Task—练习1

程序运行结果：

```
Documents/OS/Lab4 via C v17.0.0-clang
❯ ./test1
This is in the father process,here write a string to the pipe.
This is in the child process,here read a string from the pipe.
Hello world , this is write by pipe.
```

分析：

该程序使用管道进行父进程和子进程之间的通信，父进程先通过管道写端向管道中写入数据，之后关闭管道的两个端，子进程之后通过管道读端从管道中读取数据，然后关闭管道的两个端。管道是半双工的，数据只能从写端流向读端，并且由于管道具有阻塞的特性，当管道为空时读操作会阻塞，当管道已满时写操作会阻塞，这就保证了车给女婿按照正确的顺序执行，不会出现调度混乱的情况。

Task—练习2

写出的程序如下：

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#include <string.h>

int main(void)
{
    int fd[2];
    char buf[100];
    pid_t pid;

    // 创建管道
    pipe(fd);

    if ((pid = fork()) > 0)  // 父进程
    {
        printf("This is in the parent process, waiting for message from child.\n");

        // 关闭写端（父进程只读）
        close(fd[1]);

        // 从管道读取数据
        int nbytes = read(fd[0], buf, sizeof(buf));
```

```
24
25          if (nbytes > 0)
26          {
27              buf[nbytes] = '\0';  // 确保字符串正确终止
28              printf("Parent received message: %s\n", buf);
29          }
30          else
31          {
32              printf("Parent: No message received.\n");
33          }
34
35          // 关闭读端
36          close(fd[0]);
37
38          // 等待子进程结束
39          waitpid(pid, NULL, 0);
40      }
41      else // 子进程
42      {
43          printf("This is in the child process, sending message to parent.\n");
44
45          // 关闭读端（子进程只写）
46          close(fd[0]);
47
48          // 准备要发送的消息
49          char message[] = "it is sending a message to parent!";
50
51          // 向管道写入数据
52          write(fd[1], message, strlen(message) + 1);  // +1 包含字符串结束符
53
54          printf("Child sent message: %s\n", message);
55
56          // 关闭写端
57          close(fd[1]);
58      }
59      return 0;
60  }
```

通过使用管道，实现父进程虽然先被创建，但是由于它要实现的操作是从管道中读取数据，所以即使先被OS调度，也会由于管道为空而被阻塞，最终先执行后被创建的执行写操作的子进程。

运行结果如下：

```
Documents/OS/Lab4 via C v17.0.0-clang
❯ ./test2
This is in the parent process, waiting for message from child.
This is in the child process, sending message to parent.
Child sent message: it is sending a message to parent!
Parent received message: it is sending a message to parent!
```

分析：

观察结果可以发现，确实是父进程先被调度，但是由于此时管道中内容为空，所以父进程被阻塞了，子进程开始执行，子进程向管道中写完数据执行完毕之后，父进程从阻塞状态恢复到执行状态继续执行，最终得到预期的结果。

四、心得总结

通过本次实验，我对 `Linux` 进程控制和管道通信机制有了更深入的理解。实验让我认识到进程与程序之间的本质区别，程序是静态的代码集合，而进程是动态执行的实体，拥有独立的资源空间。通过 `pipe()` 函数创建管道，我直观地理解了进程间通信的基本原理，特别是管道作为半双工通信机制的特点，数据只能单向流动，这要求父子进程必须明确各自的读写角色。实验中遇到的阻塞现象也让我体会到操作系统调度的重要性，当管道为空时读操作会阻塞，这保证了进程执行的正确顺序，避免了数据竞争问题。

实践过程中，我掌握了 `fork()` 创建子进程、 `pipe()` 建立通信管道、 `read()` / `write()` 进行数据读写以及 `waitpid()` 实现进程同步等关键系统调用的使用。特别是文件描述符的管理让我意识到系统资源清理的重要性，正确关闭不需要的文件描述符是编写健壮程序的基本要求。这次实验不仅加深了我对进程通信机制的理论认识，更锻炼了我解决实际编程问题的能力，让我体会到操作系统底层机制对上层应用开发的支撑作用，为今后深入学习并发编程和系统级开发打下了坚实基础。