

电子科技大学

实验报告

学生姓名：Lolipop 学号：2018091202000 指导教师：xx

实验地点：校外 实验时间：2020/06/09

一、实验名称：利用管道实现两个进程的通信

二、实验学时：4 学时

三、实验目的：了解进程间通信的原理。

四、实验原理：

- 1) 每个进程各自有不同的用户地址空间
- 2) 进程之间要交换数据必须通过内核（内核开辟的缓冲区）
- 3) 内核提供的这种机制称为进程间通信（IPC, InterProcess Communication）
- 4) 管道是一种最基本的 IPC 机制，由 pipe 函数创建：

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

- a) 返回值：成功返回 0，错误返回-1
- b) 参数 fd 为管道描述符，同文件描述符，可以使用文件 I/O 函数（close, read, write）进行操作。
- c) fd[0]为管道读端；fd[1]为管道写端。如不需要，可以关闭相应端的描述符。

五、实验内容：

- 1) 在 Linux 系统中使用系统调用 fork() 创建两个子进程，
- 2) 使用系统调用 pipe() 建立一个管道，两个子进程分别向管道各写一句话：

```
Child process 1 is sending a message!  
Child process 2 is sending a message!
```

- 3) 父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。然后分别结束两个子进程的运行。

六、实验器材（设备、元器件）：

- 1) 硬件：一台 Windows 10 电脑；
- 2) 软件：XShell。

七、实验步骤：

- 1) 创建一个管道。
- 2) 创建子进程 1，向管道中写入 “Child process 1 is sending a message!”，并做好跟父进程的同步执行。
- 3) 创建子进程 2，向管道中写入 “Child process 2 is sending a message!”，并做好跟父进程的同步执行。
- 4) 父进程从管道中读取数据，并打印输出。

八、实验结果与分析（含重要数据结果分析或核心代码流程分析）

- 1) 创建一个管道，如代码 8-1 所示。

代码 8-1 创建管道

```
int fd[2];  
if (pipe(fd) == -1) { // 管道创建失败  
    return -1;  
}
```

- 2) 创建子进程 1，向管道中写入 “Child process 1 is sending a message!”，并做好跟父进程的同步执行，如代码 8-2 所示。

代码 8-2 创建子进程 1

```
if ((pid_1 = fork()) == 0) { // 子进程 1 创建成功  
    close(fd[0]); // 关闭管道读端  
    sprintf(w_buf, "Child process 1 is sending a message!\n");  
    len_1 = strlen(w_buf);  
    write(fd[1], w_buf, len_1);  
    exit(0);  
}
```

```
}
```

- 3) 创建子进程 2，向管道中写入 “Child process 2 is sending a message!”，并做好跟父进程的同步执行，如代码 8-3 所示。

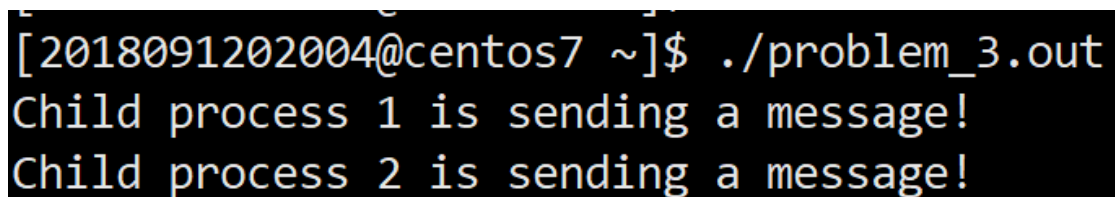
代码 8-3 创建子进程 2

```
waitpid(pid_1, NULL, 0); // 等待子进程 1 处理完毕
if ((pid_2 = fork()) == 0) { // 子进程 2 创建成功
    close(fd[0]);
    sprintf(w_buf, "Child process 2 is sending a message!\n");
    len_2 = strlen(w_buf);
    write(fd[1], w_buf, len_2);
    exit(0);
}
```

- 4) 父进程从管道中读取数据，如代码 8-4 所示；并打印输出，如图 8-1 所示。

代码 8-4 处理父进程

```
waitpid(pid_2, NULL, 0); // 等待子进程 2 处理完毕
close(fd[1]); // 关闭管道写端
if(read(fd[0], r_buf, len_1) > 0)
    printf("%s\n", r_buf);
if(read(fd[0], r_buf, len_2) > 0)
    printf("%s\n", r_buf);
exit(0);
```



```
[2018091202004@centos7 ~]$ ./problem_3.out
Child process 1 is sending a message!
Child process 2 is sending a message!
```

图 8-1 管道通信实验结果

- 5) 实现管道通信（多子进程）的代码截图如代码 8-5 所示。

代码 8-5 管道通信问题代码

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

```
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main (void) {
    pid_t pid_1, pid_2; // 两个子进程
    int fd[2];
    char w_buf[100], r_buf[100]; // 读写数据流
    int len_1, len_2;

    if (pipe(fd) == -1) { // 管道创建失败
        return -1;
    } else { // 管道创建成功
        if ((pid_1 = fork()) == 0) { // 子进程 1 创建成功
            close(fd[0]); // 关闭管道读端
            sprintf(w_buf, "Child process 1 is sending a message!\n");
            len_1 = strlen(w_buf);
            write(fd[1], w_buf, len_1);
            exit(0);
        } else if (pid_1 > 0) {
            waitpid(pid_1, NULL, 0); // 等待子进程 1 处理完毕
            if ((pid_2 = fork()) == 0) { // 子进程 2 创建成功
                close(fd[0]);
                sprintf(w_buf, "Child process 2 is sending a message!\n");
                len_2 = strlen(w_buf);
                write(fd[1], w_buf, len_2);
                exit(0);
            } else if (pid_2 > 0) { // 处理父进程
                waitpid(pid_2, NULL, 0); // 等待子进程 2 处理完毕
                close(fd[1]); // 关闭管道写端
                if (read(fd[0], r_buf, len_1) > 0)
                    printf("%s\n", r_buf);
            }
        }
    }
}
```

```
        if(read(fd[0], r_buf, len_2) > 0)
            printf("%s\n", r_buf);
        exit(0);
    }
}

return 0;
}
```

报告评分:

指导教师签字: