

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 操作系统基础

理论教师 任立勇

实验教师 任立勇

电子科技大学

实验报告

学生姓名：袁昊男 学号：2018091618008 指导教师：任立勇

实验地点：在线实验 实验时间：2020.05.27

一、实验室名称：信息与软件工程学院实验中心

二、实验名称：利用管道实现两个进程的通信

三、实验学时：4 学时

四、实验原理：

首先创建两个子进程，注意 Linux 下使用 `fork()` 函数创建进程的方法。

父进程和两个子进程间需要同步，使用 `waitpid()` 函数实现父进程等待子进程运行完毕后从管道中读取数据并打印，只有子进程将数据写入管道后，父进程才能够执行打开管道操作。

由于 `fork` 函数让子进程完整地拷贝了父进程的整个地址空间，所以子进程都有管道的读端和写端。所以在相关进程中最好关掉不用的一端。

根据要求，“父进程先接收子进程 P1 发来的消息，然后再接收子进程 P2 发来的消息。”存在两个同步问题，两个子进程和父进程之间（先子写后父读）同步、子进程 1 和子进程 2 之间（先 1 写，再 2 写）

五、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。

六、实验内容：

在 Linux 系统中使用系统调用 `fork()` 创建两个子进程，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!

要求:

- (1) 父进程先接收子进程 P1 发来的消息, 然后再接收子进程 P2 发来的消息。
- (2) 在 Linux 平台下实现。

七、实验器材 (设备、元器件):

- 1、学生每人一台 PC, 安装 WindowsXP/2000 操作系统。
- 2、局域网络环境。
- 3、个人 PC 安装 VMware 虚拟机和 Ubuntu 系统。

八、实验步骤:

- 1、使用 “Ctrl+Alt+T” 打开终端。
- 2、使用 gedit 或 vim 命令打开文本编辑器进行编码: “gedit 文件名.c”。
- 3、编译程序:

gcc 文件名.c -o 可执行程序名

(如果只输入 “gcc 文件名.c”, 默认可执行程序名为 a.out)

使用线程库时, gcc 编译需要添加 -lpthread。

- 4、执行程序: ./可执行程序名。

九、实验数据及结果分析

1、代码

```
1. #include <unistd.h>
2. #include <signal.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/wait.h>
8.
9. int main()
10. {
11.     int pid1, pid2;
12.     int fd[2];                //管道描述符
13.     char outpipe[100], inpipe[100]; //存储字符串
14.     pipe(fd);                //创建管道
15.
16.     while((pid1 = fork()) == -1) ; //父进程中返回子进程 id, 子进程
    中返回 0
17.     if(pid1==0)                //子进程
18.     {
19.         close(fd[0]);          //关闭读端
20.         lockf(fd[1],1,0);      //lockf()函数允许将文件区域用作
    信号量,1 互斥锁定区域,0 锁定字节数
```

```

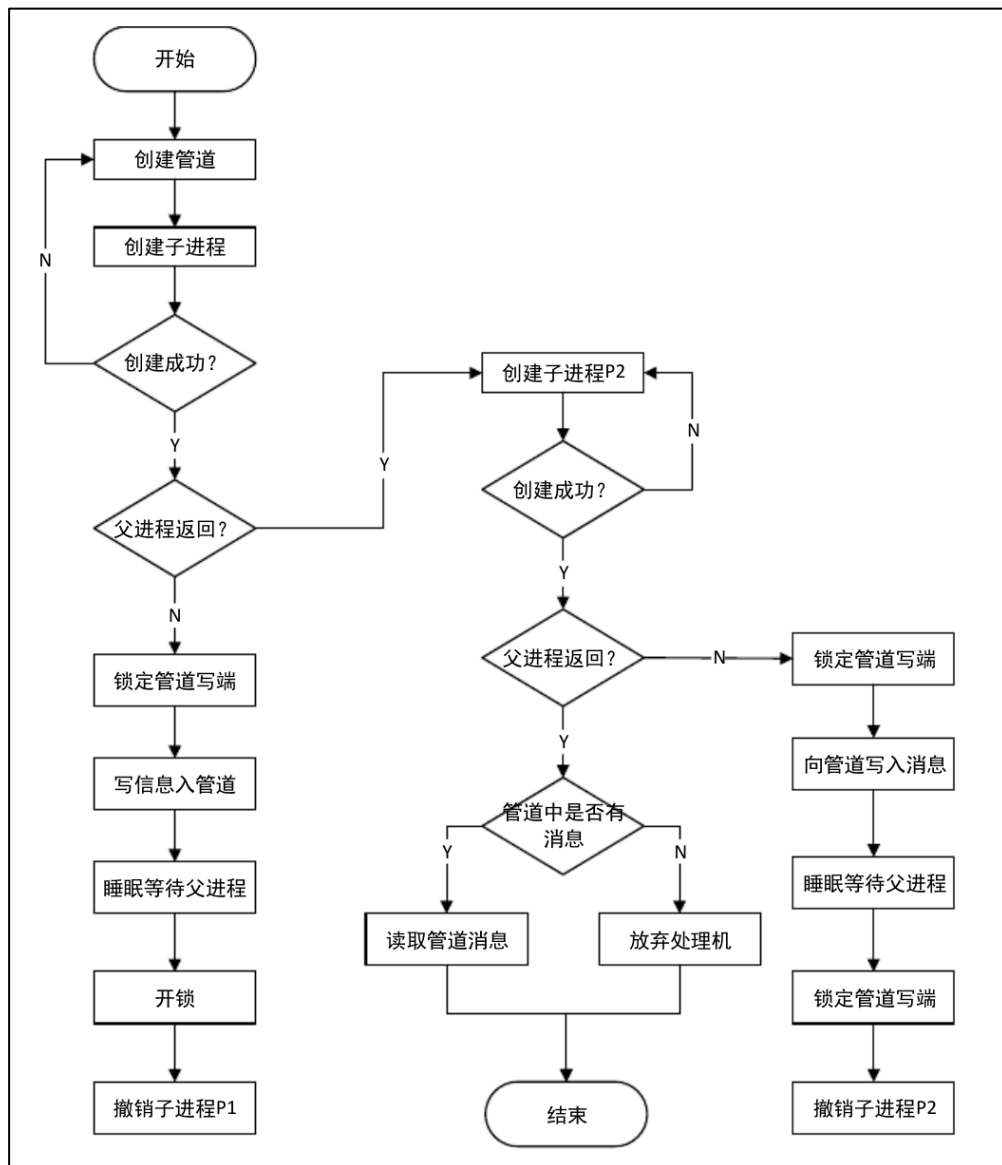
21.     sprintf(outpipe, "Child process 1 is sending a message!");
22.     // printf("p1\n");
23.     write(fd[1],outpipe,50);           //向管道写
24.     sleep(3);
25.     lockf(fd[1],0,0);                 //0 解锁互斥区域
26.     exit(0);
27. }
28. else //父进程返回
29. {
30.     while((pid2 = fork()) == -1) ; //出错返回-1
31.     if(pid2==0)
32.     {
33.         sleep(1);                     //先阻塞，使 pid1 优先执行
34.         close(fd[0]);
35.         lockf(fd[1],1,0);
36.         sprintf(outpipe,"Child process 2 is sending a message!");
37.         // printf("p2\n");
38.         write(fd[1],outpipe,50);
39.         sleep(3);
40.         lockf(fd[1],0,0);
41.         exit(0);
42.     }
43.     else
44.     {
45.         waitpid(0, NULL, 0);          //等待当前所有子进程结束
46.         close(fd[1]);                 //关闭写端
47.         read(fd[0],inpipe,50);        //从管道读
48.         printf("%s\n",inpipe);
49.
50.         waitpid(0, NULL, 0);
51.         read(fd[0],inpipe,50);
52.         printf("%s\n",inpipe);
53.         exit(0);
54.     }
55. }
56. }

```

说明：主要分为父进程操作与子进程操作。

- 在父进程中，首先调用 `pipe()`函数创建管道 `fd`，再调用 `fork()`函数创建子进程 1 与子进程 2。子进程成功创建后调用 `waitpid()`函数等待从 `fd` 管道中读出子进程 1、2 写入的字符串，并显示在屏幕上。
- 在子进程中，首先将 `fd` 管道写入端 (`fd[1]`) 加上互斥锁，防止其他子进程同时写入数据。互斥锁上锁后，将待写入的字符串写入字符串数组 `OutPipe` 暂存，再调用 `write()`函数将 `OutPipe` 中指定字节数的信息写入管道中。此后调用 `sleep()`函数自我阻塞，等待父进程从 `fd` 管道中读出数据，之后将写入端互斥锁解锁，退出。
- `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，避免不必要的错误产生。

2、程序流程图



3、运行结果截图

```
yhn@yhn-virtual-machine: /mnt/hgfs/Network Programming
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
yhn@yhn-virtual-machine: /mnt/hgfs/Network Programming$ ./3.out
Child process 1 is sending a message!
Child process 2 is sending a message!
yhn@yhn-virtual-machine: /mnt/hgfs/Network Programming$
```

十、实验结论

从实验运行结果中可以看出，父进程成功从管道中读取子进程写入管道的两个字符串，并按照要求的顺序输出。

十一、总结及心得体会

解决父、子进程之间同步问题的关键是确定管道中数据是否存在。

在本实验中，父进程创建子进程 P1→子进程 P1 执行→返回父进程执行，并创建子进程 P2→子进程 P2 执行→返回父进程执行→程序结束。父进程在从管道读出数据之前必须先确定管道中有数据，若管道中没有数据，则父进程调用 `wait()` 函数将自身阻塞。只有子进程结束后、管道中数据已经存在，父进程才会执行。同理，子进程在向管道写入数据之前要确定管道中的数据已经被父进程读出，否则将调用 `wait()` 函数阻塞自己。

此外，在子程序对管道进行操作时，必须对管道加上互斥锁，防止另一个子程序同时向管道中写入数据，造成此前已经写入的数据被覆盖。

通过本实验，我对操作系统中管道机制以及父子进程同步问题有了更深刻的理解。

十二、对本实验过程及方法、手段的改进建议

本实验设计与教材结合紧密、难度适中，通过对操作系统中管道机制解决父子进程间通信问题的实现，强化了学生对有关父子进程问题、管道机制、临界区资源互斥访问等知识点的理解与掌握。此外，还使学生熟悉 Linux 系统的基本环境，了解 Linux 下进程和线程的实现，对操作系统上的深入学习打下了坚实的基础。

报告评分：

指导教师签字：

附：选做实验代码及运行结果截图

1、代码

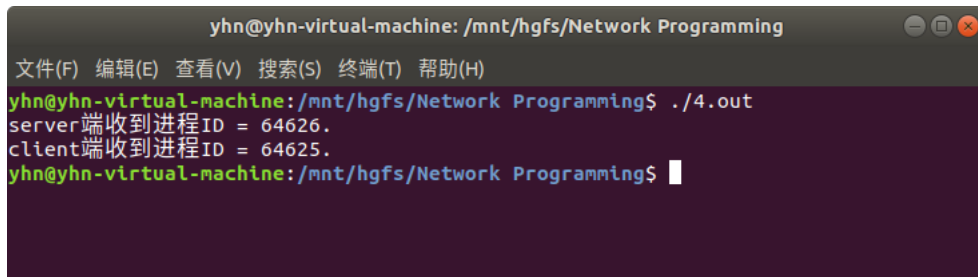
```
1. #include <sys/types.h>
2. #include <sys/msg.h>
3. #include <sys/ipc.h>
4. #include <stdlib.h>
5. #include <stdio.h>
6. #include <sys/types.h>
7. #include <sys/wait.h>
8. #include <unistd.h>
9.
10. struct msgform
11. {
12.     long mtype;
13.     char mtext[250];
14. }msg;
15.
16. long REQ = 1;
17. int msgqid, pid, *pint;
18.
19. void server()
20. {
21.     msgqid = msgget(1234,0666|IPC_CREAT); //创建消息队列, 0666 表示对内存有
        读 / 写 / 执行的权限, 返回队列 ID
22.     msgrcv(msgqid,&msg,250,REQ,0); //接收 client 的消息
23.     pint = (int *)msg.mtext;
24.     pid = *pint; //获取 client 的 pid
25.     printf("server 端收到进程 ID = %d.\n",pid);
26.     msg.mtype = pid; //消息类型为 client 的 pid
27.     *pint = getpid(); //将消息内容设置为 server 的 id
28.     msgsnd(msgqid,&msg,250,0);
29.     exit(0);
30. }
31.
32. void client()
33. {
34.     msgqid = msgget(1234,0666); //打开消息队列, 0666 表示对内存有
        读 / 写 / 执行的权限
35.     pid = getpid(); //得到自己的 pid
36.     pint = (int *)msg.mtext; //指向消息正文
37.     *pint = pid; //将正文改为自己的 pid 值
38.     msg.mtype = REQ; //设置消息类型
39.     msgsnd(msgqid,&msg,250,0); //发送消息, 成功 0, 错误-1
40.     msgrcv(msgqid,&msg,250,pid,0); //接受 server 发的消息
41.     printf("client 端收到进程 ID = %d.\n", *pint);
42.     exit(0);
43. }
44.
45. int main()
46. {
47.     int pid1 = fork(); //创建进程
48.     if(!pid1) server();
49.
50.     int pid2 = fork();
51.     if(!pid2) client();
52.
```

```
53.     wait(0);  
54.     wait(0);  
55. }
```

说明：主要分为服务器端函数 `server()` 与客户端函数 `client()`。

- 在 `server()` 函数中，首先调用 `msgget()` 函数以指定 `key` 值创建队列，函数返回队列 ID，然后调用 `msgrcv()` 函数接收来自客户端的 REQ 类型的消息，获取到客户端消息中传递的客户端进程号并显示在屏幕上。同时，将服务器端的进程号写入消息结构体中，传递给客户端进程。
- 在 `client()` 函数中，首先打开执行 `key` 值的消息队列，将客户端进程号写入消息结构体中，调用 `msgsnd()` 函数发送。然后调用 `msgrcv()` 函数接收来自服务器端的消息，获取到服务器端消息中传递的服务器端进程号并显示在屏幕上。
- 在 `main()` 函数中，调用 `fork()` 函数分别创建服务器端进程与客户端进程。之后调用 `wait()` 函数将自我阻塞。

2、运行结果截图



```
yhn@yhn-virtual-machine: /mnt/hgfs/Network Programming  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
yhn@yhn-virtual-machine:/mnt/hgfs/Network Programming$ ./4.out  
server端收到进程ID = 64626.  
client端收到进程ID = 64625.  
yhn@yhn-virtual-machine:/mnt/hgfs/Network Programming$
```