



南昌大学实验报告

学生姓名： 丁俊 学 号： 8003119100 专业班级： 信安 193
班

实验类型： ☐ 验证 ☐ 综合 ☐ 设计 ☐ 创新 实验日期： 4.29 实验成绩： _____

一、 实验项目名称

进程间通信

二、 实验目的

- 1、了解什么是信号。
- 2、熟悉 LINUX 系统中进程之间软中断通信的基本原理。
- 3、了解什么是管道。
- 4、熟悉 UNIX/LINUX 支持的管道通信方式。

三、 实验基本原理

信号量机制实验

1、编写一段程序，使用系统调用 `fork()` 创建两个子进程，再用系统调用 `signal()` 让父进程捕捉键盘上来的中断信号（即按 `ctrl+c` 键），当捕捉到中断信号后，父进程用系统调用 `kill()` 向两个子进程发出信号，子进程捕捉到信号后，分别输出下列信息后终止： `Child process 1 is killed by parent!` `Child process 2 is killed by parent!` 父进程等待两个子进程终止后，输出以下信息后终止： `Parent process is killed!`

程序见实验步骤

实验要求：

- (1)、运行程序并分析结果。
- (2)、如果把 `signal(SIGINT,stop)` 放在①号和②号位置，结果会怎样并分析原因。
- (3)、该程序段前面部分用了两个 `wait(0)`,为什么？
- (4)、该程序段中每个进程退出时都用了语句 `exit(0)`,为什么？

2、司机售票员问题（选做题） 编程用 `fork()` 创建一个子进程代表售票员，司机在父进程中，再用系统调用 `signal()` 让父进程（司机）捕捉来自子进程（售票员）发出的中断信号，让子进程（售票员）捕捉来自（司机）发出的中断信号，以实现进程间的同步运行。

进程管道通信实验

1、编制一段程序，实现进程的管道通信。使用 `pipe()` 建立一条管道线。两个子进程 `p1` 和 `p2` 分别向管道各写一句话： `Child 1 is sending message!` `Child 2 is sending message!` 而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。

程序见到实验步骤

实验要求：

- 1、运行程序并分析结果。
- 2、在父进程中用 `pipe()` 建立一条管道线，往管道里写一句话，两个子进程接收这句话。

四、 主要仪器设备及耗材

PC+linux 系统

GCC

五、 实验步骤

信号量机制实验

1.实验分析

代码

```
1. # include<stdio.h>
2. # include<signal.h>
3. # include<unistd.h> // 创建进程
4. # include<stdlib.h> // 等待进程
5. void waiting();
6. void stop();
7. int wait_mark;
8. int main()
9. {   int p1, p2;
10.    signal(SIGINT,stop);
11.    while((p1=fork())!=-1); // 创建子进程，不成功为-1，然后一直创建
12.    if(p1>0) // 此时是父进程，再次创建子进程
13.    {
```

```

14.     while((p2=fork())!=-1); // 创建子进程，不成功为-1
15.     if(p2>0) // 父进程
16.     {
17.         wait_mark=1;
18.         waiting(0);
19.         kill(p1,10);
20.         kill(p2,12);
21.         wait(0);
22.         wait(0); // 两个 wait 等待两个子进程的结束
23.         printf("parent process is killed!\n");
24.         exit(0);
25.     }
26.     else // 现在在第二个子进程
27.     {
28.         wait_mark=1;
29.         signal(12,stop);
30.         waiting();
31.         lockf(1,1,0);
32.         printf("child process 2 is intereupted by parent!\n");
33.         lockf(1,0,0); // 打开输出锁
34.         exit(0); // 返回值，与前面的 wait 相对应
35.     }
36. }
37. else
38. {
39.     wait_mark=1;
40.     signal(10,stop);
41.     waiting();
42.     lockf(1,1,0); // 输出锁，锁定屏幕输出，输出 printf
43.     printf("child process 1 is killed by parent!\n");
44.     lockf(1,0,0);
45.     exit(0);
46. }
47. }
48. void waiting()
49. {
50.     while(wait_mark!=0);
51.     // 调用 waiting，变量不为 1，符合条件就开始循环检测，不管哪个进程被调度都会什么
    都不出现，
52.     // 因为都在执行 waiting 函数，所有设置了下面的 stop 函数
53. }
54. void stop()
55. {
56.     wait_mark=0; // 当父进程阻塞时，调度子进程进行，执行结果会再次进入父进程

```

运行结果

```
ncu@ncu-virtual-machine: ~/DingJun/work2
ncu@ncu-virtual-machine:~/DingJun/work2$ gcc commun.c -o a1.out
ncu@ncu-virtual-machine:~/DingJun/work2$ ./a1.out
^Cchild process 2 is intereupter by parent!
child process 1 is killed by parent!
parent process is killed!
ncu@ncu-virtual-machine:~/DingJun/work2$
```

如果把 `signal(SIGINT,stop)`放在 1 位置，运行结果：

```
ncu@ncu-virtual-machine: ~/DingJun/work2
ncu@ncu-virtual-machine:~/DingJun/work2$ gcc commun.c -o a2.out
ncu@ncu-virtual-machine:~/DingJun/work2$ ./a2.out
^?^Cchild process 2 is intereupter by parent!
parent process is killed!
ncu@ncu-virtual-machine:~/DingJun/work2$
ncu@ncu-virtual-machine:~/DingJun/work2$ ./a2.out
^Cchild process 2 is intereupter by parent!
parent process is killed!
ncu@ncu-virtual-machine:~/DingJun/work2$
```

如果放在 2 号位置，运行结果：

```
ncu@ncu-virtual-machine: ~/DingJun/work2
ncu@ncu-virtual-machine:~/DingJun/work2$ gcc commun.c -o a3.out
ncu@ncu-virtual-machine:~/DingJun/work2$ ./a3.out
^Cparent process is killed!
ncu@ncu-virtual-machine:~/DingJun/work2$
```

实验要求

- (1) 分析：程序首先声明了控制进程软中断信号 `signal(SIGINT,stop)`，当运行程序之后按下`^c`时就会触发软中断，调用 `stop` 函数将 `wait_mark` 变为 0。因为 `signal` 是声明在程序的全局部分，所以所有包括子进程、父进程的进程都会受到影响，输出自己当前进程结束的语句，当两个子进程全部结束后，父进程也随之结束并输出结束语句。
- (2) 现象：如果把 `signal(SIGINT,stop)`放在 1 号位置，没有出现进程 1；如果放在

2 号位置，只出现父进程。

原因：因为 `signal` 函数在哪声明在哪起作用。当在全局部分时，`waiting` 函数等待 `wait_mark` 为 0 时才输入进程结束提示，而此时对所有的进程都有影响。当在不同的进程中声明的作用域不同。比如在 `if(p2>0)` 语句块中，只能输出父进程结束的语句。

(3) 两个 `wait()` 是等待两个子进程结束。

(4) `exit(0)` 是返回值，0 就是返回的参数，可以用来判断函数是否正确返回。

2、司机售票员问题

代码

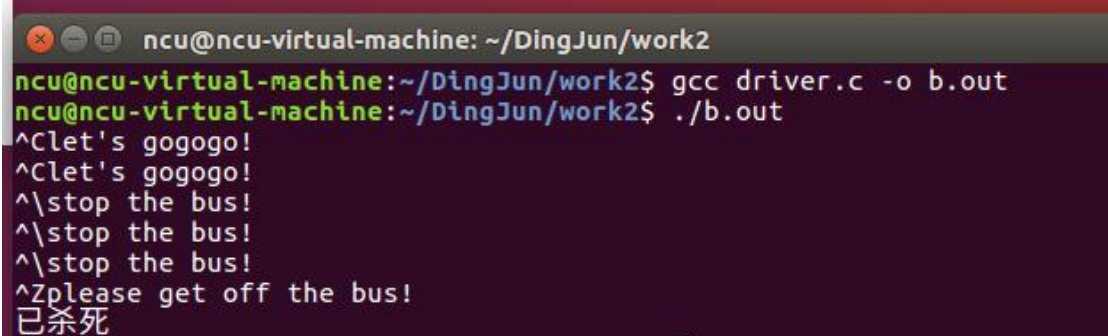
```
1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <signal.h>
4. #include <unistd.h>
5. void saler(int);    //售票员信号处理函数
6. void driver(int);  //司机信号处理函数
7. pid_t pid; //保存子进程号
8. int main()
9. {
10. if ((pid = fork()) == -1)
11. {
12.     perror("fork");
13.     return -1;
14. }
15. if (pid == 0) //子进程（售票员）
16. {
17.     signal(SIGINT, saler);    //处理 SIGINT 信号，不退出进程（如果不注册该信号，则会本进程退出）。注：ctrl+c 可向所有进程发送退出信号，默认退出所有进程
18.     signal(SIGQUIT, saler);  //处理 SIGQUIT 信号，不退出进程。注：ctrl+\ 可向所有进程发送退出信号，默认退出所有进程
19.     signal(SIGUSR1, saler); //处理用户自定义信号 SIGUSR1
20.     signal(SIGTSTP, SIG_IGN); //忽略 SIGTSTP 信号。注：ctrl+z 可向所有进程发送该信号，暂停所有进程
21.     // SIG_IGN 表示信号被忽略
22.     // SIGTSTP 使进程暂停
23.     while (1)
24.         pause(); // pause 函数。调用该函数可以造成进程主动挂起，等待信号唤醒
25. }
26. else //父进程（司机）
27. {
28.     signal(SIGINT, SIG_IGN);
29.     signal(SIGQUIT, SIG_IGN);
```

```

30.     signal(SIGTSTP, driver);
31.     signal(SIGUSR1, driver);
32.     signal(SIGUSR2, driver);
33.     while (1)
34.         pause();
35. }
36. return 0;
37. }
38.
39. void saler(int n)
40. {
41.     if (n == SIGINT)
42.         kill(getppid(), SIGUSR1); //如果接收到 SIGINT 信号，则向父进程（司机）发
        送 SIGUSR1 信号
43.     if (n == SIGQUIT)
44.         kill(getppid(), SIGUSR2);
45.     if (n == SIGUSR1)
46.     {
47.         printf("please get off the bus!\n");
48.         kill(0, SIGKILL); //杀死该进程组的所有进程
49.     }
50. }
51.
52. void driver(int n)
53. {
54.     if (n == SIGUSR1)
55.         printf("let's gogogo!\n");
56.     if (n == SIGUSR2)
57.         printf("stop the bus!\n");
58.     if (n == SIGTSTP)
59.         kill(pid, SIGUSR1); //如果接受到 SIGTSTP 信号，则向子进程（售票员）发送
        SIGUSR1 信号
60. }

```

运行结果



```

ncu@ncu-virtual-machine: ~/DingJun/work2
ncu@ncu-virtual-machine:~/DingJun/work2$ gcc driver.c -o b.out
ncu@ncu-virtual-machine:~/DingJun/work2$ ./b.out
^Clet's gogogo!
^Clet's gogogo!
^C\stop the bus!
^C\stop the bus!
^C\stop the bus!
^Cplease get off the bus!
已杀死

```

结果分析

在子进程中，处理 SIGINT 信号，按下 ^C 时，使用 kill(getppid, SIGUSR1) 向父进程(司机)发送 SIGUSR1(用户自定义)信号，driver 函数中当收到 SIGUSR1 信号时输出 "let's gogogo!\n"。

当输入 ^\ 时，同理子进程向父进程传递中断消息，当司机收到 SIGUSR2(用户自定义)信号输出 "stop the bus!\n"。

当输入 ^Z 时，在 driver 函数中当收到暂停信号 SIGSTP 时，使用 kill(pid, SIGUSR1) 向子进程(售票员)发送 SIGUSR1 信号，此时 saler 函数中接收到该信号，就输出 "please get off the bus!\n"。

进程管道通信实验

代码

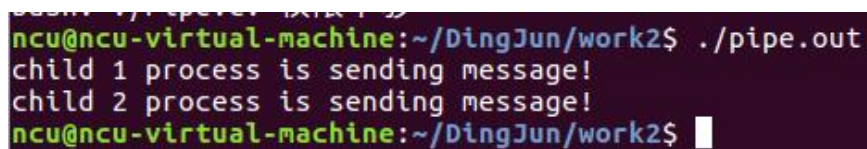
```
1. #include<unistd.h>
2. #include<signal.h>
3. #include<stdio.h>
4. #include<stdlib.h>
5. int pid1,pid2;
6. int main()
7. {
8.     int fd[2];
9.     char OutPipe[100],InPipe[100];
10.    pipe(fd); // 创建管道，fd[0]指向管道的读端，fd[1]指向管道的写端
11.    while((pid1=fork())== -1);
12.    if(pid1==0)
13.    {
14.        lockf(fd[1],1,0);
15.        sprintf(OutPipe, "child 1 process is sending message!");
16.        write(fd[1],OutPipe,50);
17.        sleep(5);
18.        lockf(fd[1],0,0);
19.        exit(0);
20.    }
21.    else
22.    {
23.        while((pid2=fork())== -1);
```

```

24.     if(pid2==0){
25.         lockf(fd[1],1,0);
26.         sprintf(OutPipe, " child 2 process is sending message!");
27.         write(fd[1],OutPipe,50);
28.         sleep(5);
29.         lockf(fd[1],0,0);
30.         exit(0);
31.     }
32.     else
33.     {
34.         wait(0);
35.         read(fd[0],InPipe,50);
36.         printf("%s\n",InPipe);
37.         wait(0);
38.         read(fd[0],InPipe,50);
39.         printf("%s\n",InPipe);
40.         exit(0);
41.     }
42. }
43. }

```

运行结果



```

ncu@ncu-virtual-machine:~/DingJun/work2$ ./pipe.out
child 1 process is sending message!
child 2 process is sending message!
ncu@ncu-virtual-machine:~/DingJun/work2$

```

分析

1、当 `pid1==0` 时，说明此时执行的是子进程 1，通过 `sprintf` 把进程运行语句写进 `OutPipe` 数组中，`fd[0]` 指向管道的读端，`fd[1]` 指向管道的写端，此时子进程 1 通过 `write` 函数把语句写入 `OutPipe` 管道的写端等待父进程读出。这里的 `lockf` 函数是把屏幕输出锁住以便实现两个子进程的互斥写入数据。

同理当在父进程里面再次创建一个子进程 2，同样地向管道写入端写子进程 2 的数据。等又一次到了父进程调用 `read` 函数把 `fd[0]` 读入端的数据读入到 `InPipe` 数组中，输出数据，最后出现两个子进程依次输出运行信息的结果。

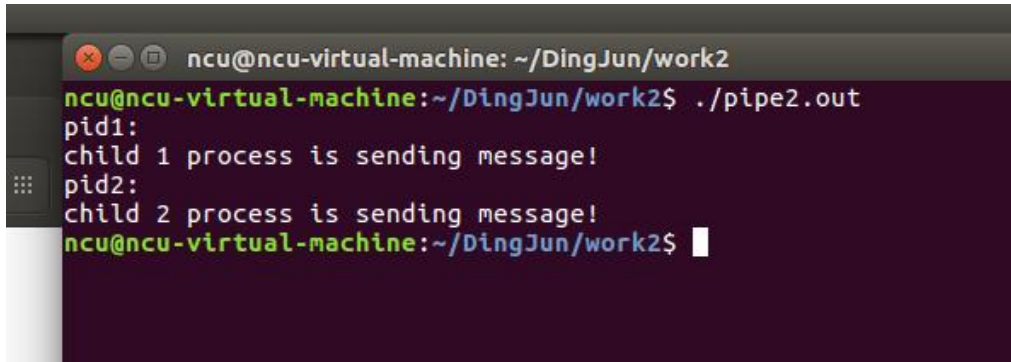
2、在父进程中用 `pipe()` 建立一条管道线，往管道里写一句话，两个子进程接收这句话。

代码

```
1. #include<unistd.h>
2. #include<signal.h>
3. #include<stdio.h>
4. #include<stdlib.h>
5. #include<sys/wait.h>
6. int pid1,pid2;
7. int main()
8. {
9.     int fd[2];
10.    char OutPipe[100],OutPipe1[100],InPipe[100];
11.    pipe(fd);
12.    while((pid1=fork())!=-1);
13.    if(pid1==0)
14.    {
15.        // wait(0);
16.        read(fd[0],InPipe,50);
17.        printf("pid1:\n%s\n",InPipe);
18.        exit(0);
19.    }
20.    else
21.    {
22.        while((pid2=fork())!=-1);
23.        if(pid2==0)
24.        {
25.            wait(0);
26.            read(fd[0],InPipe,50);
27.            printf("pid2:\n%s\n",InPipe);
28.            exit(0);
29.        }
30.        else
31.        {
32.            lockf(fd[1],1,0);
33.            sprintf(OutPipe,"child 1 process is sending message!");
34.            write(fd[1],OutPipe,50);
35.            sleep(1);
36.            lockf(fd[1],0,0);
37.            sprintf(OutPipe1,"child 2 process is sending message!");
```

```
40.     write(fd[1],OutPipe1,50);
41.     exit(0);
42.
43.     }
44. }
45. }
```

结果



```
ncu@ncu-virtual-machine: ~/DingJun/work2
ncu@ncu-virtual-machine:~/DingJun/work2$ ./pipe2.out
pid1:
child 1 process is sending message!
pid2:
child 2 process is sending message!
ncu@ncu-virtual-machine:~/DingJun/work2$
```

通过在父进程写入两个子进程运行的信息,在 `fd[1]` 写入端写入 `OutPipe` 数据,然后分别在两个子进程中通过 `fd[0]` 读入端读出进程运行信息即可。

六、 思考讨论题或体会或对改进实验的建议

通过这次实验我学会了如何通过管道和信号量机制来实现进程之间的通信和交流

七、 参考资料