实验四　进程通信

学号：19020011038　　姓名：岳宇轩　　年级：2019

一．PIPE 系统调用的基本使用方法

1.具体要求与步骤

1) 编写一 C/C++语言程序（程序名为 basicpipe.c/basicpipe.cpp），使用系统调用 pipe()实现基本的消息传递。
2) 多次连续反复运行这个程序，观察屏幕显示结果，试简单分析其原因。
3) 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码 1 所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出 19os 的式样。

2. 实验结果截图

结果分析：

程序中有两个进程：父进程和子进程。

子进程从管道的 **filedes[0]**端读出缓冲区大小的内容，写入到缓冲区中并且调用屏幕打印 **IO** 进行输出；

父进程先管道 **filedes[1]**端写入一个字符串；

经过多次的尝试，程序只有结果所示的一种结果顺序，即：父进程先获得处理机，向管道写入了一个字符串；子进程获得处理机后从管道读出字符串到缓冲区并且输出；结果如图 **2** 所示。

还有另外一种输出顺序，我在父进程输出“`This is in the father process,here write a string to the pipe`”之前加一句 **sleep(2)**，模仿子进程先获得处理机的场景。结果是：子进程先获得处理机资源，

想从管道中读取数据，结果管道中还没有数据，故要将自己阻塞。等待父进程 **2s sleep** 之后完成管道数据写入操作，子进程被唤醒，读出数据并且向屏幕输出。结果如图 **3.**

```c
*This file is created by:                                                     *
*      #     #############        ######            #######        *
*      #     #            #      #        #         #        #       *
*      #     #            #    #            #       #          #     *
*      #     #            #    #            #       #                *
*      #     #            #    #            #          #             *
*      #     #############   #            #          ##              *
*      #                  #    #            #              ##        *
*      #                  #    #            #                 #      *
*      #                  #    #            #                  #     *
*      #                  #    #            #      #           #     *
*      #                  #     #        #          #          #     *
*      #     #############        ######            #######        *
*********************************************************************/

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

int main(void)
{
    int filedes[2];
    char buf[80];
    int pid_t,pid;
    pipe( filedes ); // set pipe read and write
    if ( (pid=fork()) == 0 )  // child peocess executing
    {
        printf("This is in the child process,here read a string from the pipe.\n" );
        read(filedes[0], buf, sizeof(buf));  // read str from pipe
        printf("%s\n", buf);  // output the str
        close(filedes[0]);  // close pipe read
        close(filedes[1]);  // close pipe write
    }
    else
    {
        // father process executing
        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( filedes[1], s, sizeof(s) );  // write str to pipe
        close( filedes[0] );  // close pipe read
        close( filedes[1] );  // close pipe write
    }
    waitpid(pid,NULL,0);
    return 0;
}
```

图 1 pipe 基本使用代码

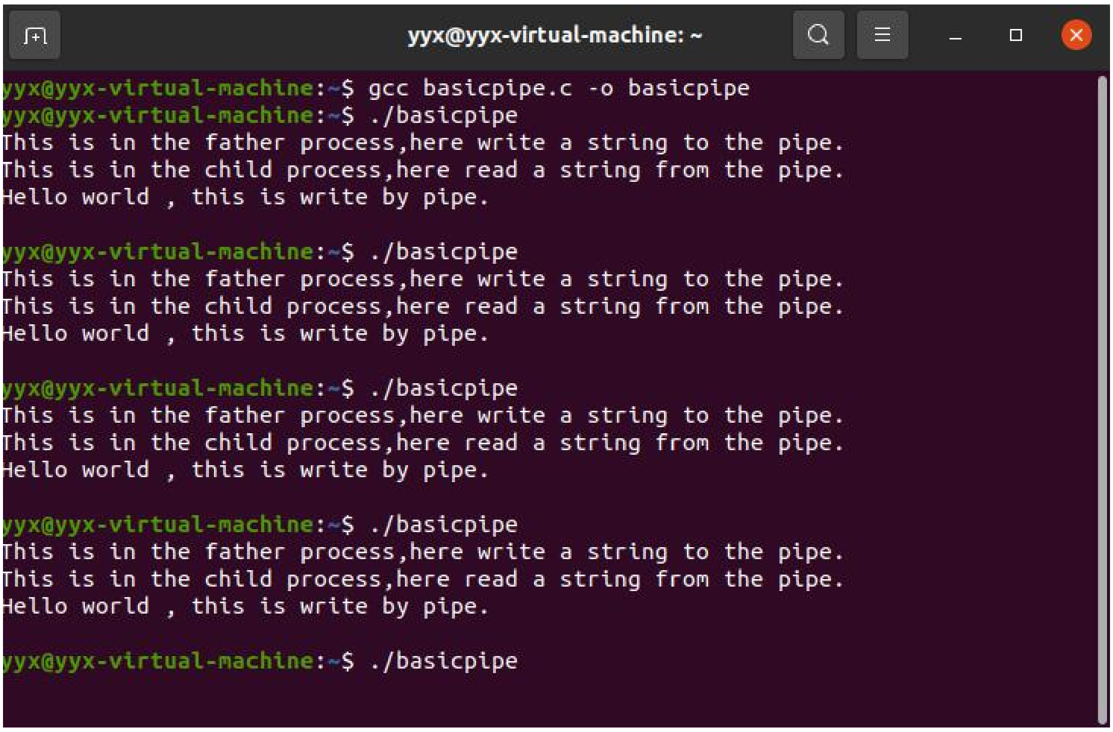

图 2 pipe 基本使用结果截图

.图 3 pipe 基本使用输出情况 2

二. PIPE 实现进程间通信

1.具体要求与步骤

1) 编写一 C/C++语言程序（程序名为 testpipe.c/testpipe.cpp），使用系统调

用 pipe( )尝试在多个进程间进行通信。

2) 要求实现的功能是，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“[进程名称] 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。多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其原因。
3) 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码 1 所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出 19os 的式样。

3. 实验结果截图

实验结果分析：

程序中共有三个进程，分别是父进程，子进程 1 和子进程 2.

子进程 1：首先将一组字符串“Child Process 1 is sending a message!**”送入缓冲区 S，然后从 S 中读出数据，通过管道的 filedes[1]句柄写入管道，随后 sleep 3s。**

子进程 2：和子进程 1 唯一的不同是写入字符串不同，子进程 2 写入的字符串是“Child Process 2 is sending a message!**”。**

父进程：父进程连续两次从管道的 filedes[0]句柄中读数据到缓冲区中并且向屏幕输出。在多次执行过程中，由于两个子进程获取处理机的先后顺序可能不同，管道中两个字符串的顺序可能不同，故父进程输出来自两个子进程通信的信息的顺序也可能不同。

打开(O) testpipe.c

```
 5 *     #     #             #       #        #         #       #      *
 6 *     #     #             #     #            #       #         #    *
 7 *     #     #             #     #            #       #              *
 8 *     #     #             #     #            #         #            *
 9 *     #     ##############      #            #          ##          *
10 *     #                   #     #            #             ##       *
11 *     #                   #     #            #               #      *
12 *     #                   #     #            #                 #    *
13 *     #                   #     #            #      #          #    *
14 *     #                   #       #        #         #        #    *
15 *     #     ##############        ######              #######       *
16 **********************************************************************/
17
18 #include <sys/types.h>
19 #include <sys/wait.h>
20 # include <stdio.h>
21 # include <unistd.h>
22 int main()
23 {
24   int i,j,fd[2];
25   char S[100];
26   pipe(fd);
27   if (i=fork()==0)  // child process 1 executing
28   {
29     sprintf(S,"Child Process 1 is sending a message!\n");  // write str into S
30     write(fd[1],S,50);  // write S into pipe with filedes[1]
31     sleep(3);  // sleep for 3s
32     return 0;
33   }
34     if (j=fork()==0)  // child process 2 executing
35   {
36     sprintf(S,"Child Process 2 is sending a message!\n");  // write str into S
37     write(fd[1],S,50);  // write S into pipe with filedes[1]
38     sleep(3);  // sleep for 3s
39     return 0;
40   }
41   else
42   {
43     wait(0);
44     read(fd[0],S,50);  // read from pipe with filedes[0] to S
45     printf("%s",S);  // output S
46     read(fd[0],S,50);  // read from pipe with filedes[0] to S
47     printf("%s",S);  // output S
48     return 0;
49   }
50 }
```

图 4 进程通信代码

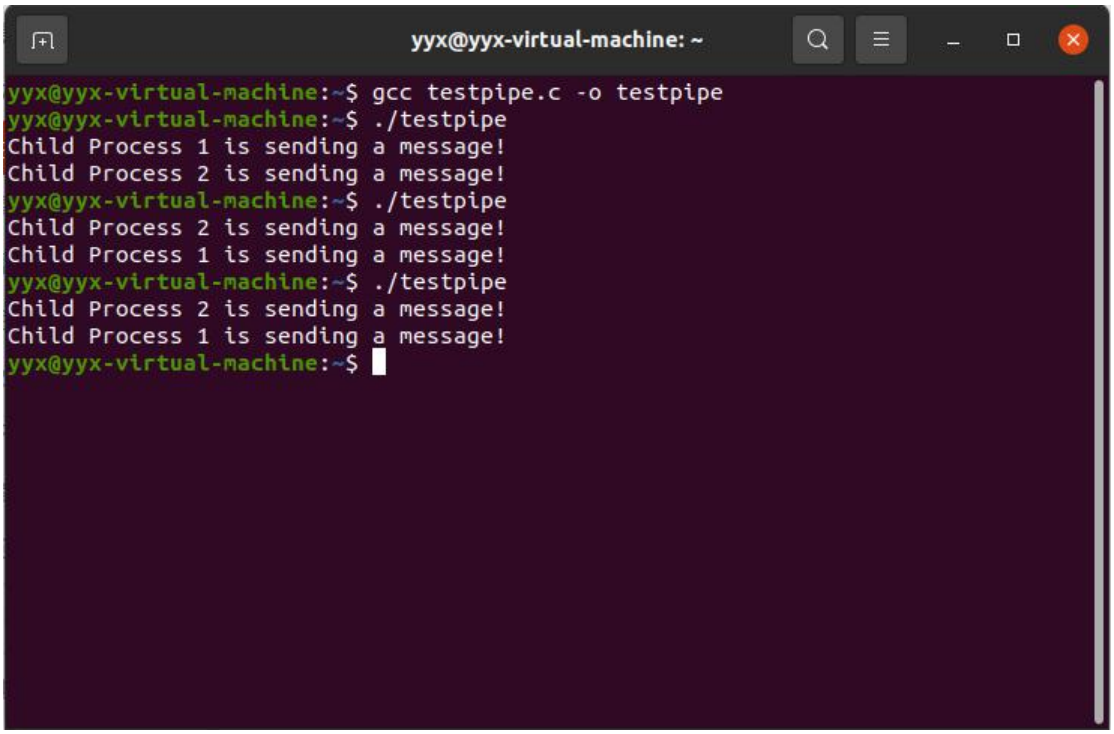
yyx@yyx-virtual-machine: ~
yyx@yyx-virtual-machine:~$ gcc testpipe.c -o testpipe
yyx@yyx-virtual-machine:~$ ./testpipe
Child Process 1 is sending a message!
Child Process 2 is sending a message!
yyx@yyx-virtual-machine:~$ ./testpipe
Child Process 2 is sending a message!
Child Process 1 is sending a message!
yyx@yyx-virtual-machine:~$ ./testpipe
Child Process 2 is sending a message!
Child Process 1 is sending a message!
yyx@yyx-virtual-machine:~$

图 5 进程通信实验结果