

浙大城市学院实验报告

- 课程名称：操作系统原理实验
- 实验项目名称：实验十 进程通信——管道
- 学生姓名：徐彬涵
- 专业班级：软件工程2003
- 学号：32001272
- 实验成绩：
- 指导老师：胡隽
- 日期：2022/05/25

实验目的

1. 了解Linux系统的进程间通信机构（IPC）；
2. 理解Linux关于管道的概念；
3. 掌握Linux支持管道的系统调用和管道的使用
4. 巩固进程同步概念

实验步骤

p1.c

程序先用pipe创建了管道，接着用fork创建了新进程。如果fork操作成功，父进程用write函数把数据写到管道中，而子进程用read函数从管道中读出数据

```
Terminal
[bex@XuBinHan_Tue May 24_18:11_~/Documents/0sLabs/0sLab10/src]$gcc -o p1 p1.c
[bex@XuBinHan_Tue May 24_18:11_~/Documents/0sLabs/0sLab10/src]$./p1
input a string:xbh
child:Read 3 bytes: xbh
parent:Wrote 3 bytes
[bex@XuBinHan_Tue May 24_18:12_~/Documents/0sLabs/0sLab10/src]$
```

p2.c

程序先用 `popen` 创建了管道，并启动了 `sort` 进程。如果管道创建成功，将乱序数据写到管道中。`sort` 进程将对输入的数据排序。

另一个创建管道的简单方法是使用库函数 `FILE *popen(char *command, char *type)`。`popen` 库函数允许一个程序把另一个程序当作一个新的进程来启动，并能对它发送数据或者

接受数据。`popen` 库函数通过在系统内部调用 `pipe()` 来创建一个 [半双工](#) 的管道，然后它创建一个子进程，启动 `shell`，最后在 `shell` 上执行 `command` 参数中的命令。管道中数据流的方向

是由第二个参数 `type` 控制的。此参数可以是 `r` 或者 `w`，分别代表读或写。但不能同时为读和写。如果成功，函数返回一个新的文件流。如果无法创建进程或者管道，返回 `NULL`。

使用 `popen()` 创建的管道必须使用 `pclose()` 关闭。`pclose` 函数等待 `popen` 进程启动的进程运行结束才关闭文件流。

```
Terminal
[bex@XuBinHan_Tue May 24_18:28_~/Documents/0sLabs/0sLab10/src]$gcc -o p2 p2.c
[bex@XuBinHan_Tue May 24_18:31_~/Documents/0sLabs/0sLab10/src]$./p2
alpha
bravo
charlie
delta
echo
[bex@XuBinHan_Tue May 24_18:31_~/Documents/0sLabs/0sLab10/src]$
```

p4.c

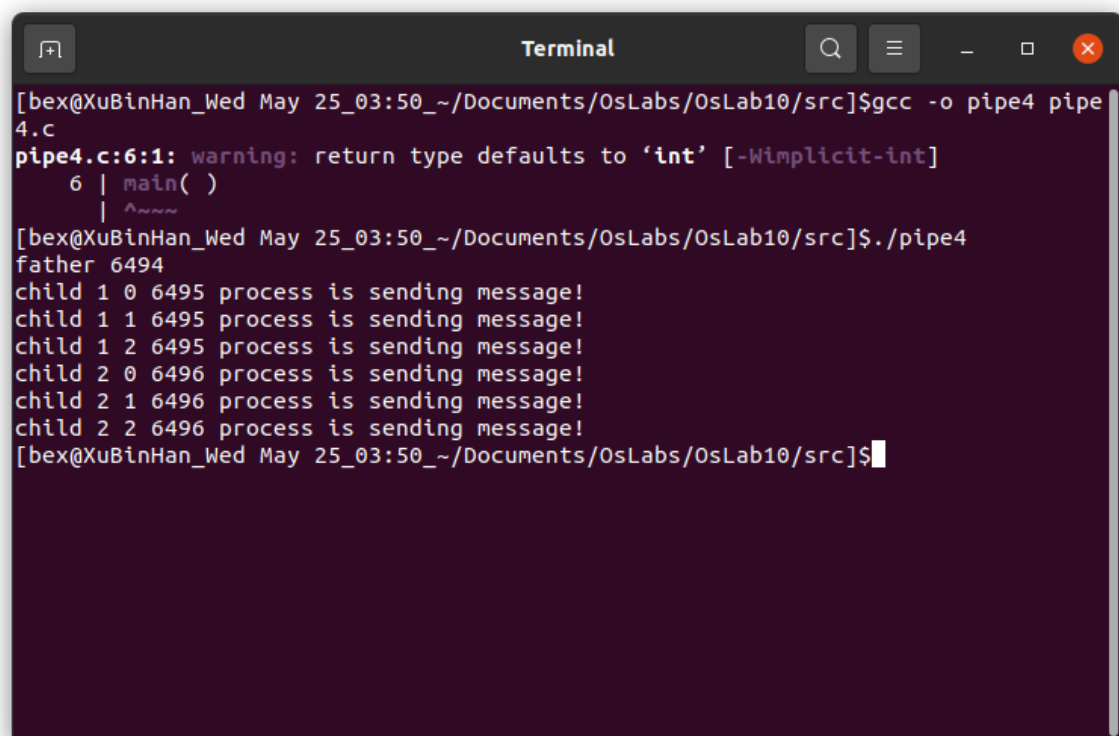
父进程关闭管道的两端，父进程不使用管道，并使用wait()等待两个子进程结束。

```
Terminal
[bex@XuBinHan_Wed May 25_03:09_~/Documents/0sLabs/0sLab10/src]$gcc -o p4 p4.c
[bex@XuBinHan_Wed May 25_03:09_~/Documents/0sLabs/0sLab10/src]$./p4
|-gnome-keyring-d --daemonize --login
| | | |-grep login
|-systemd-logind
[bex@XuBinHan_Wed May 25_03:10_~/Documents/0sLabs/0sLab10/src]$
```

父进程创建的第一个子进程关闭了管道的fd1并关闭了默认的stdout输出口，然后调用dup将fd0设为输出口，最后再将与管道连接的fd0副本删除，只留下了一个fd0的输出口，父进程的第二个子进程关闭了管道的fd0并关闭了默认的stdin输入口，然后调用dup将fd1设为输入口，最后再将与管道连接的fd1副本删除，只留下了一个fd1的输入口，第一个子进程执行 `pstree -a` 后将输出通过管道传给第二个子进程当作输入，第二个子进程再对这个输入执行 `grep login` 操作,由于第二个进程本身的输出口没变，仍为stdout，从而将结果输出

整体运行结果相当于执行指令 `pstree -a | grep login`

pipe4.c



```
[bex@XuBinHan_Wed May 25_03:50_~/Documents/OsLabs/OsLab10/src]$ gcc -o pipe4 pipe4.c
pipe4.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
    6 | main( )
      | ^~~~~~
[bex@XuBinHan_Wed May 25_03:50_~/Documents/OsLabs/OsLab10/src]$ ./pipe4
father 6494
child 1 0 6495 process is sending message!
child 1 1 6495 process is sending message!
child 1 2 6495 process is sending message!
child 2 0 6496 process is sending message!
child 2 1 6496 process is sending message!
child 2 2 6496 process is sending message!
[bex@XuBinHan_Wed May 25_03:50_~/Documents/OsLabs/OsLab10/src]$
```

父进程创建了两个子进程，在第一个子进程被创建后，同时调用 `lockf()` 函数将管道设为互斥锁定区域，这时候另一个子进程由于也调用了 `lockf()` 就会先等待第一个进程将内容全部写入调用 `lockf()` 将管道解锁后才可以写入，从而实现了进程间对于写入管道的互斥，最后父进程在子进程结束后将管道内的内容全部输出

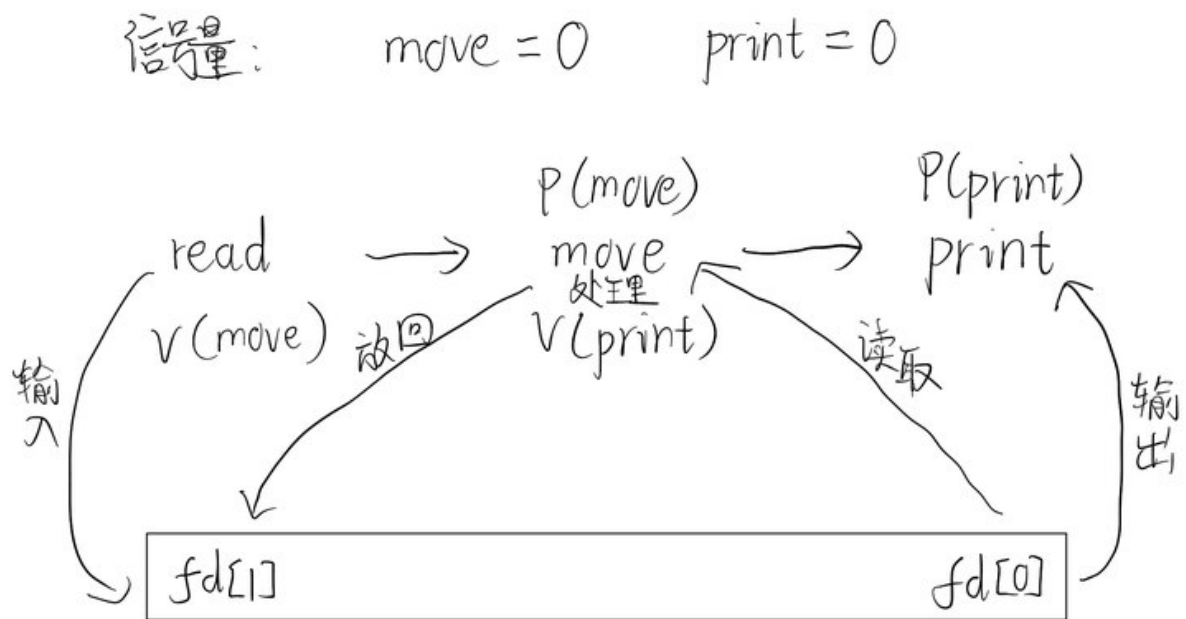
编程题

假定系统有三个并发进程read，move和print共享缓冲器B1和B2。进程read负责从输入设备上读信息，每读出一个记录后把它存放到缓冲器B1中。进程move从缓冲器B1中取出一个记录，加工后存入缓冲器B2。进程print将B2中的记录取出打印输出。缓冲器B1和B2每次只能存放一个记录。要求三个进程协调完成任务，使打印出来的与读入的记录个数，次序完全一样。试创建三个进程，用 `pipe()` 打开两个管道，如下

图所

示，实现三个进程之间的同步。

使用一个管道和信号量实现通信



```
1  #include <errno.h>
2  #include <fcntl.h>
3  #include <semaphore.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/ipc.h>
8  #include <sys/sem.h>
9  #include <sys/types.h>
10 #include <sys/wait.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 #define STR_MAX_SIZE 100
15 #define CHECK(x) \
16     do { \
17         if (!(x)) { \
18             fprintf(stderr, "%s:%d: ", __func__, __LINE__); \
19             perror(#x); \
20             exit(-1); \
21         } \
22     } while (0)
```

```

23
24
25 int main() {
26     int fd[2], pid, i = 0, j = 0;
27     int len;
28     int count;
29     ssize_t n;
30     char str[STR_MAX_SIZE];
31     //sem_t *read_mutex;
32     sem_t *move_mutex;
33     sem_t *print_mutex;
34     //read_mutex = sem_open("pipe_test_read", O_CREAT | O_RDWR, 0666, 1);
35     move_mutex = sem_open("pipe_test_move", O_CREAT | O_RDWR, 0666, 0);
36     print_mutex = sem_open("pipe_test_print", O_CREAT | O_RDWR, 0666, 0);
37
38     printf("plz tell me the quantity: ");
39     scanf("%d", &count);
40
41     CHECK(pipe(fd) >= 0);
42     CHECK((pid = fork()) >= 0);
43
44     if (pid == 0) { // child1 read
45         //sem_wait(read_mutex);
46
47         for(i = 1; i <= count; ++i){
48             printf("plz input the %d string: ", i);
49             scanf("%s", str);
50             write(fd[1], str, STR_MAX_SIZE);
51         }
52
53         sem_post(move_mutex);
54         exit(EXIT_SUCCESS);
55     }
56
57     CHECK((pid = fork()) >= 0);
58     if (pid == 0) { //child2 move
59         sem_wait(move_mutex);
60
61         for(i = 1; i <= count; ++i){
62             read(fd[0], str, STR_MAX_SIZE);
63             printf("move received the %d string %s, and move it\n", i, str);
64             for(j = 0; j < strlen(str); ++j){

```

```

65         if(str[j] >= 97 && str[j] <= 122)
66             str[j] -= 32;
67     }
68     write(fd[1], str, STR_MAX_SIZE);
69 }
70
71     sem_post(print_mutex);
72     exit(EXIT_SUCCESS);
73 }
74
75     CHECK((pid = fork()) >= 0);
76     if (pid == 0) { //child3 print
77         sem_wait(print_mutex);
78
79         for(i = 1; i <= count; ++i){
80             read(fd[0], str, STR_MAX_SIZE);
81             printf("print received the %d string %s\n", i, str);
82         }
83
84         exit(EXIT_SUCCESS);
85     }
86
87     wait(0);
88     wait(0);
89     wait(0);
90
91     printf("Here is parent , my children are over\n");
92
93     //sem_close(read_mutex);
94     sem_close(move_mutex);
95     sem_close(print_mutex);
96     //sem_unlink("pipe_test_read");
97     sem_unlink("pipe_test_move");
98     sem_unlink("pipe_test_print");
99     return 0;
100 }

```

```
Terminal
[bex@XuBinHan_Wed May 25_17:56_~/Documents/OsLabs/OsLab10/src]$gcc -o MyCode MyCode.c -lpthread
[bex@XuBinHan_Wed May 25_17:56_~/Documents/OsLabs/OsLab10/src]$./MyCode
plz tell me the quantity: 3
plz input the 1 string: 123
plz input the 2 string: bex
plz input the 3 string: 233{abcdefg}
move received the 1 string 123, and move it
move received the 2 string bex, and move it
move received the 3 string 233{abcdefg}, and move it
print received the 1 string 123
print received the 2 string BEX
print received the 3 string 233{ABCDEFG}
Here is parent , my children are over
[bex@XuBinHan_Wed May 25_17:56_~/Documents/OsLabs/OsLab10/src]$
```