

**操作系统原理实验报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓 名： | | 吴阳民 | | |
| **学 院：** | | U201614515 | | |
| 专 业： | | 计算机科学与技术专业 | | |
| 班 级： | | 1601 | | |
| 学 号： | | U201614515 | | |
| 指导教师： | | 2018年12月15日 | | |
|  |  | |  |  |
| 分数 | |  |
| 教师 | |  |
|  | |  |
| 2018年12月15日 | | | | |

目 录

[1. 实验一 进程控制 1](#_Toc534951231)

[1.1. 实验目的 1](#_Toc534951232)

[1.2. 实验内容 1](#_Toc534951233)

[1.3. 实验设计 1](#_Toc534951234)

[1.3.1. 开发环境 1](#_Toc534951235)

[1.3.2. 实验设计 1](#_Toc534951236)

[1.3.3. 实验步骤 3](#_Toc534951237)

[1.3.4. 实验调试及心得 4](#_Toc534951238)

[2. 实验二 线程同步与通信 5](#_Toc534951239)

[2.1. 实验目的 5](#_Toc534951240)

[2.2. 实验内容 5](#_Toc534951241)

[2.3. 实验设计 5](#_Toc534951242)

[2.3.1. 开发环境 5](#_Toc534951243)

[2.3.2. 实验设计 5](#_Toc534951244)

[2.4. 实验调试 5](#_Toc534951245)

[2.4.1. 实验步骤 6](#_Toc534951246)

[2.4.2. 实验调试及心得 6](#_Toc534951247)

[3. 实验三 内存共享与进程同步 7](#_Toc534951248)

[3.1. 实验目的 7](#_Toc534951249)

[3.2. 实验内容 7](#_Toc534951250)

[3.3. 实验设计 7](#_Toc534951251)

[3.3.1. 开发环境 7](#_Toc534951252)

[3.3.2. 实验设计 7](#_Toc534951253)

[3.4. 实验调试 8](#_Toc534951254)

[3.4.1. 实验步骤 8](#_Toc534951255)

[3.4.2. 实验调试及心得 8](#_Toc534951256)

[4. 实验四 Linux文件目录 9](#_Toc534951257)

[4.1. 实验目的 9](#_Toc534951258)

[4.2. 实验内容 9](#_Toc534951259)

[4.3. 实验设计 9](#_Toc534951260)

[4.3.1. 开发环境 9](#_Toc534951261)

[4.3.2. 实验设计 9](#_Toc534951262)

[4.4. 实验调试 9](#_Toc534951263)

[4.4.1. 实验步骤 9](#_Toc534951264)

[4.4.2. 实验调试及心得 10](#_Toc534951265)

[5. 附录 实验代码 11](#_Toc534951266)

[5.1 实验一代码 11](#_Toc534951267)

[5.1.1. 头文件lab1.h 11](#_Toc534951268)

[5.1.2. 源文件lab1.c 11](#_Toc534951269)

[5.2实验二代码 13](#_Toc534951270)

[5.1.3. 头文件lab2.cpp 13](#_Toc534951271)

[5.1.4. 源文件lab2.cpp 14](#_Toc534951272)

[5.3实验三代码 16](#_Toc534951273)

[5.1.5. 头文件header.h 16](#_Toc534951274)

[5.1.6. 源文件get.c 17](#_Toc534951275)

[5.1.7. 源文件put.c 17](#_Toc534951276)

[5.1.8. 源文件main.c 18](#_Toc534951277)

[5.4实验四代码 19](#_Toc534951278)

[5.1.9. 头文件lab4.h 19](#_Toc534951279)

[5.1.10. 源文件lab4.cpp 20](#_Toc534951280)

1. 实验一 进程控制
   1. 实验目的
2. 加深对进程的理解,进一步认识并发执行的实质；
3. 分析进程争用资源现象,学习解决进程互斥的方法；
4. 掌握Linux进程基本控制；
5. 掌握Linux系统中的软中断和管道通信。
   1. 实验内容

编写程序，演示多进程并发执行和进程软中断、管道通信。

* 父进程使用系统调用pipe( )建立一个管道,然后使用系统调用fork()创建两个子进程，子进程1和子进程2；
* 子进程1每隔1秒通过管道向子进程2发送数据:

I send you x times. (x初值为1，每次发送后做加一操作）

子进程2从管道读出信息，并显示在屏幕上。

* 父进程用系统调用signal()捕捉来自键盘的中断信号（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 l is Killed by Parent!

Child Process 2 is Killed by Parent!

* 父进程等待两个子进程终止后，释放管道并输出如下的信息后终\

Parent Process is Killed!

* 1. 实验设计
     1. 开发环境

Windows Subsystem for Linux

Visual Studio Code

gcc 6.1.0

* + 1. 实验设计

首先给出实验指导书中给出的伪代码

1. main( ) {
2. 创建无名管道；
3. 设置软中断信号SIGINT；
4. 创建子进程1、2；
5. 等待子进程1、2退出；
6. 关闭管道；
7. }
8. 父进程信号处理 {
9. 发SIGUSR1给子进程1；
10. 发SIGUSR1给子进程2；
11. }
12. 子进程1 {
13. 设置忽略信号SIGINT；
14. 设置信号SIGUSR1；
15. **while**(1) {
16. 发送数据至管道数据；
17. 计数器++；
18. 睡眠1秒；
19. }
20. }
21. 子进程2 {
22. 设置忽略信号SIGINT；
23. 设置信号SIGUSR1；
24. **while**(1) {
25. 接收管道数据；
26. 显示数据；
27. }
28. }
29. SIGUSR1信号处理 {
30. 关闭管道；
31. 显示退出信息；
32. 退出；
33. }

在以上伪代码中给出了实验流程，首先实现两个进程并发执行，并且通过无名管道通信以及实现进程的软中断信号处理机制。其中进程创建利用pid=fork（）函数，创建一个子进程，被创建的子进程是父进程的进程映像的一个副本 (除proc结构外) ；关于进程间的通信使用管道，其中管道用于具有亲缘关系的进程通信，管道为半双工，需建立两个管道，数据只能单向流动，另外对两端进程来说，管道即是存在于内存的文件；对于进程的软中断信号，即信号机制，提供一种简单的处理异步事件的方法，在一个或多个进程之间传递异步信号，当信号出现的时候，可以忽略，或捕捉信号后调用定义函数或执行系统默认动作。综合上述操作，完成本次实验要求。

* + 1. 实验步骤
    2. 创建子进程相互通信的无名管道：int piped[2];pipe(piped);其中piped[0]用于读操作，piped[1]用于写操作。
    3. 设置主进程与子进程的信号处理函数。
    4. 主进程信号处理函数：即signal(SIGINT,sign\_handler1)，主进程响应信号SIGINT，即Ctrl+C信号，执行自己定义的软中断处理程序sign\_handler1（该程序内调用kill函数，分别向两个子进程发送信号）。
    5. 子进程信号处理函数：1）设置忽略接受到主程序响应的中断Ctrl+C信号的信号处理函数，即signal(SIGINT, SIG\_IGN)，其中SIG\_IGN表示忽略相对应的信号。
    6. 设置响应由父进程传递到的信号响应函数，即signal(SIGUSR1, sign\_handler2)，signal(SIGUSR2, sign\_handler3);其中SIGUSR1 ，SIGUSR2为自己设定的来自父进程的传递信号，后者为设定的响应函数，其中exit（0），表示正常终结该进程。
    7. 创建两个子进程1,2；采取下述创建方法，创建两个父进程的兄弟进程：

1. **if**((a=fork())==0){
2. 进程1
3. }
4. **else**{
5. **if**((b=fork())== 0){
6. 进程2
7. }
8. }
   * 1. 然后对子进程1,2进行编写，根据题目要求，在子进程1中设定死循环，每隔一秒向管道进行写操作，其中n循环递加；同样地在子进程2中设定死循环，不断地从管道进行读操作，并进行打印操作。
     2. 主程序中调用waitpid()函数，结束所有子进程，然后关闭管道。
     3. 实验调试及心得

根据上述完成程序后，运行程序，执行结果如图 1 程序运行中结果图所示：

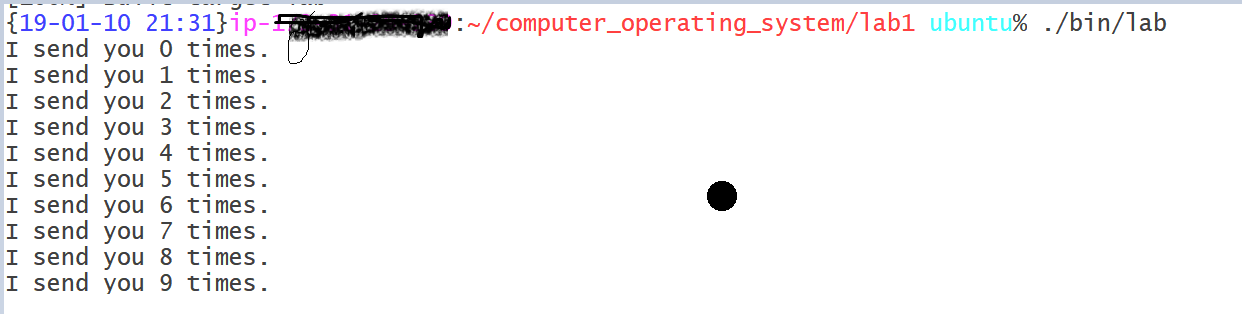


图 1 程序运行中结果图

由上图所示，进程1,与进程2先后创建成功，并调用getpid（）返回目前进程的进程标识，然后屏幕上显示进程2打印的信息，由于子进程1每次循环，等待1秒，所以在屏幕上为每隔一秒打印一次的显示结果。

现在在键盘上输入Ctrl+C，程序执行结果如图 2所示：

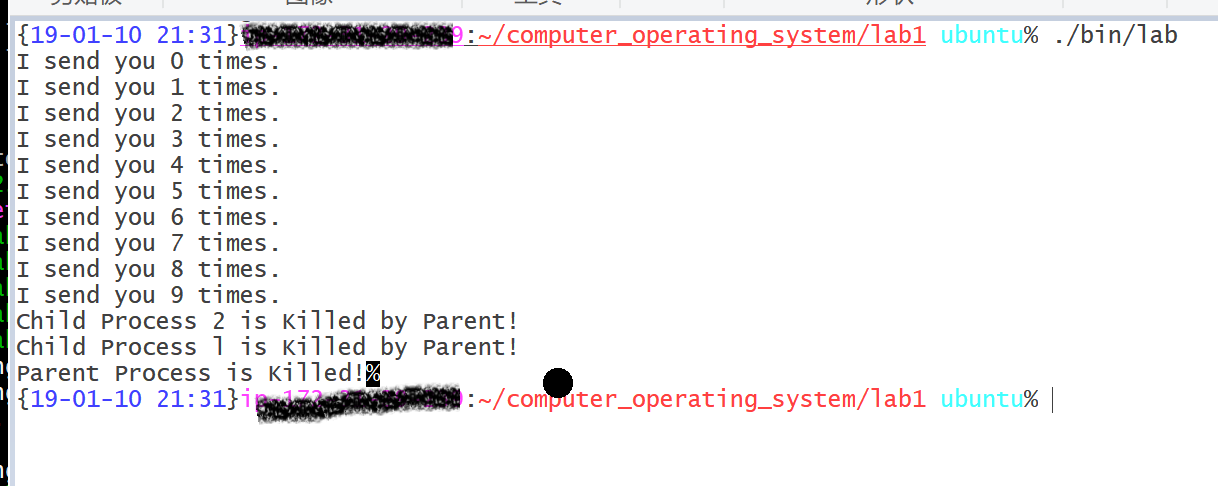


图 2程序响应信号执行结果

由上图所知，当输入Ctrl+C后，触发中断信号后屏幕上显示了子进程1,2分别被结束，然后父进程结束，结果验证正确。

**心得：**在本次进程控制实验中，学到了很多东西，对于多进程的创建有了更深入的理解，避免了创建了子进程的子进程，并且也学习到了利用管道实现进程间的通信，以及如何设置信号处理机制。综上，受益匪浅。

1. 实验二 线程同步与通信
   1. 实验目的
2. 掌握Linux下线程的概念；
3. 了解Linux线程同步与通信的主要机制；
4. 通过信号灯操作实现线程间的同步与互斥。
   1. 实验内容

通过Linux多线程与信号灯机制，设计并实现计算机线程与I/O线程共享缓冲区的同步与通信。程序要求:

两个线程,共享公共变量a

线程1负责计算(1到100的累加，每次加一个数)

线程2负责打印（输出累加的中间结果)

* 1. 实验设计
     1. 开发环境

Windows Subsystem for Linux

Visual Studio Code

gcc 6.1.0

* + 1. 实验设计

在本次实验中我们需要实现两个线程，对一个共享变量进行操作，首先需要我们了解到linux下的线程的基本操作，其次我们使用信号灯实现线程的互斥与同步，关于信号灯，在linux下中信号灯是一个数据集合，可以单独使用这一集合的每个元素，即该信号量中的元素，即是使用的信号灯，对信号灯进行P，V操作，其中P操作对信号灯值减1，若相减结果小于0，则调用进程受阻，并插入到该信号灯的等待队列中，否则可以继续执行。再V操作对信号灯值加1，若相加结果大于0，进程继续执行，否则，唤醒在信号灯等待队列上的进程。综合上述操作，完成本次实验要求。

* 1. 实验调试
     1. 实验步骤
* 创建信号量集，并且创建2个信号灯分别为0，1，并赋初值其中0号信号灯初值为1,1号信号灯初值为0
* 创建线程1,2并与程序1,2绑定。
* 书写程序1.2使其达到实验目的要求。
  + 1. 实验调试及心得

根据上述要求，完成程序设计，运行程序，得到运行如图 3所示。

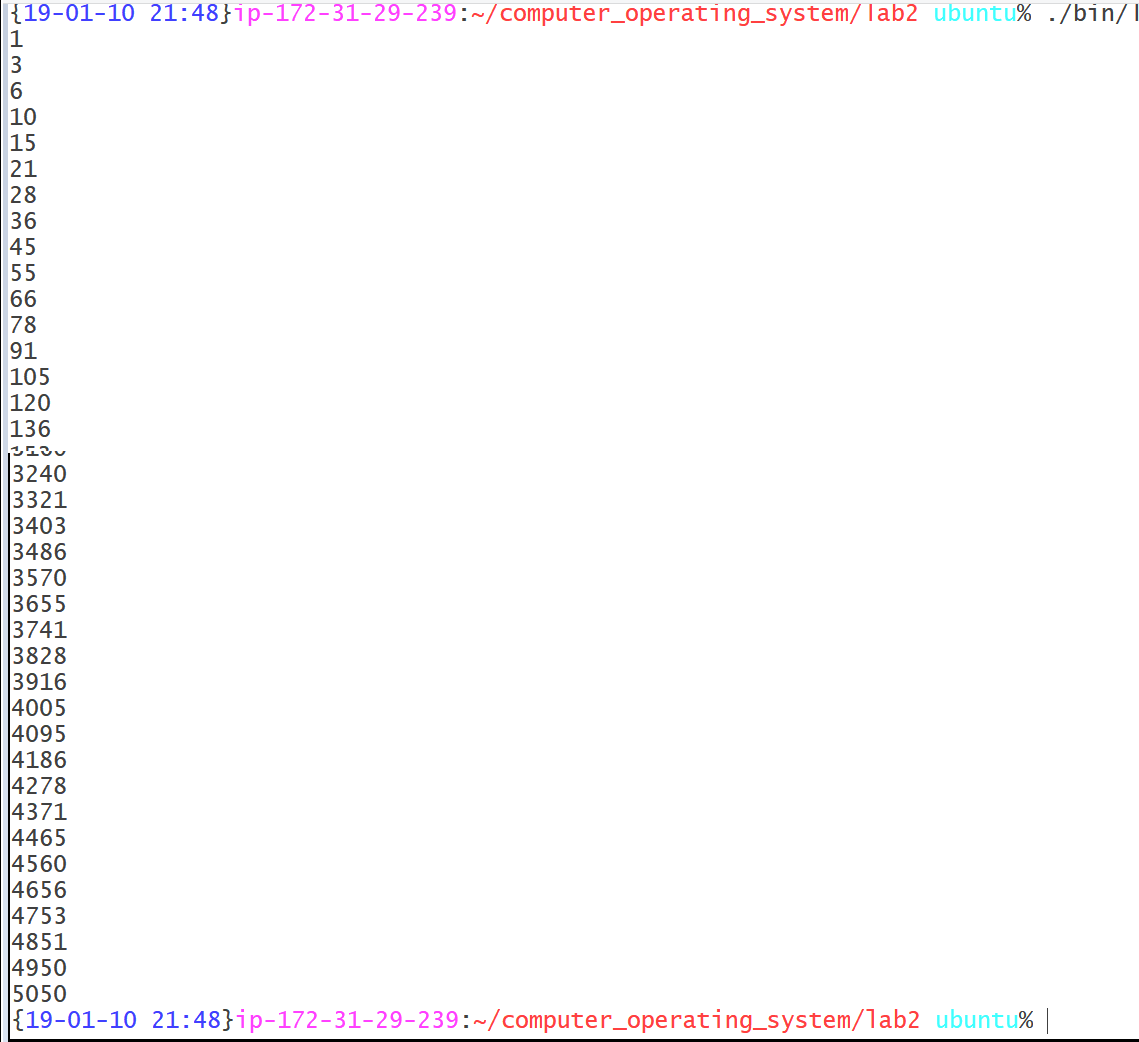


图 3

结果分析：结果如图所示通过信号灯的使用，实现了两个线程交替执行，达到实验目的。

1. 实验三 内存共享与进程同步
   1. 实验目的
2. 掌握Linux下共享内存的概念与使用方法；
3. 掌握环形缓冲的结构与使用方法；
4. 掌握Linux下进程同步与通信的主要机制。
   1. 实验内容

利用多个共享内存（有限空间）构成的环形缓冲，将源文件复制到目标文件，实现两个进程的誊抄。

* 1. 实验设计
     1. 开发环境

Windows Subsystem for Linux

Visual Studio Code

gcc 6.1.0

* + 1. 实验设计

本验三使用和实验二相同的 P，V 操作。主进程首先创建两个信号量，分别用于记录当前的空闲缓冲区和非空闲缓冲区个数。然后主进程还要创建一个环形缓冲区待使用，该环形共享缓冲区采用一个指针数组来实现，即cir\_buf \*addr[Buffernum];每个指针指向一个cir\_buf的内存空间，该内容空间包含一个指向Buffersize大小空间的int缓冲区的指针，即int \*virtaddr[Buffersize];和一个int变量readsize，用于记录每次写入缓冲区的字节数量。然后主进程创建两个子进程read\_from\_file和write\_to\_file，分别用于从源文件读数据进入缓冲区和从缓冲区读数据进入标准输出。

共享缓冲区采用指针数组的形式定义，主进程申请共享缓冲区个指针，每个指针指向一个共享缓冲区，即cir\_buf \*addr[Buffernum]，而每个共享缓冲区包括两项，每次从文件中读取数据的大小int readsize，用于最后判断文件是否读取到结尾，如果读取的数据为0.那么说明文件数据读取结束。还有是int \*virtaddr[Buffersize]，指向一个指定大小空间的内存块，用于存放每次从文件中读出的数据。通过对数组下标的循环控制，来实现对缓冲区的环形利用构建。即i =（i+1）%Buffernum。

其中read\_from\_file子进程负责将源文件里面的数据读到共享缓冲区里面，首先以只读的方式打开源文件。由于共享缓冲区是临界资源，需要进行P,V操作，严格控制对其的读写。所以需要采用信号灯，为了与写进程统一，需要和主进程，write\_to\_file进程采用同一套信号灯，因此需要使用主进程创建的信号灯。semid = semget(SHMKEY,2,IPC\_CREAT|0666)。然后为了将数据写入缓冲区，需要将主进程申请的内存空间与本进程相链接。即使用shmat函数，具体为：addr[i] = (cir\_buf\*)shmat(shmid[i],NULL,0)。在准备工作完成之后，首先对信号灯0进行P操作，判断是否具有空闲缓冲区，如果有，则从文件中读取数据，当读取的数据内容长度不为0时，将数据写入缓冲区，然后将空闲缓冲区数量减1，即P(semid，0),同时将载入数据缓冲区个数加1，即V(semid，1)，当读取数据完毕时，将载入数据缓冲区个数加1，即V(semid，1)，关闭文件，并将虚拟地址与共享缓冲区的链接断开。

write\_to\_file进程与read\_from\_file子进程的准备工作相似，只是在打开文件时，需要带有O\_CREAT和O\_TRUNC标志，在文件不存在的时候创建新文件。在文件已经存在且为普通文件，那么先清空文件。对缓冲区进行操作时，首先需要判断是否有缓冲区具有数据，即P(semid,1)，判断当前是否具有写入内容的缓冲区。如果有，而且当前的缓冲区在读文件时，数据长度不为0，即addr[i]->readsize > 0，则将该缓冲区数据写到文件里面，然后将空闲缓冲区加一V(semid,0); 然后进入下一次判断。

* 1. 实验**调试**
     1. 实验步骤

编写源码在main函数中调用，重新生成makefile，执行make命令生成程序。

* + 1. 实验调试及心得

程序运行结果如图 4所示

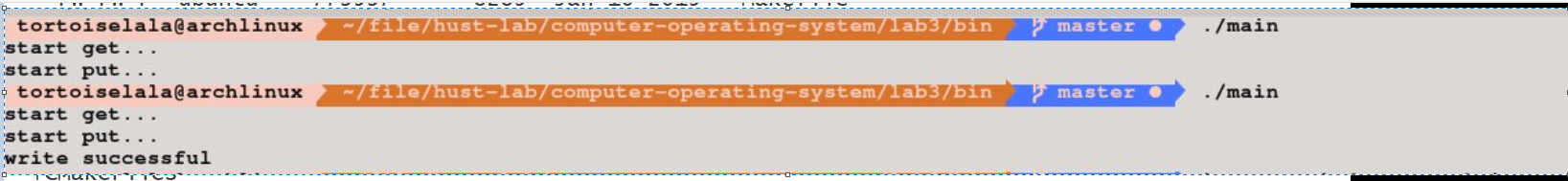


图 4 程序3执行结果

1. 实验四 Linux文件目录
   1. 实验目的
2. 了解Linux文件系统与目录操作；
3. 了解Linux文件系统目录结构；
4. 掌握文件和目录的程序设计方法。
   1. 实验内容

编程实现目录查询功能：

* 功能类似ls -lR；
* 查询指定目录下的文件及子目录信息；显示文件的类型、大小、时间等信息；
* 递归显示子目录中的所有文件信息
  1. 实验设计
     1. 开发环境

Windows Subsystem for Linux

Visual Studio Code

gcc 6.1.0

* + 1. 实验设计

由于实验要求采用深度优先算法对文件目录进行递归打印，同时由于需要打印文件路径名，为了区分不同级文件夹下面的文件，对其进行不同程度的缩进表示。即具体为：对不同目录级下面的文件，采用先打印文件路径名，然后如果是子目录，则打印信息比上一级内容向后缩进一个空格，然后开始递归打印该文件夹下的文件信息，传入的参数为当前的文件夹路径和深度。如果已经是文件，则直接打印输出文件的具体信息即可。Printfdir函数具体的算法流程图如图4-1所示。对于具体每一项文件信息的获取，可以通过lstat函数，以文件夹名获取文件夹信息，并保存在buf所指的结构体中。

* 1. 实验调试
     1. 实验步骤

编写源码在main函数中调用，重新生成makefile，执行make命令生成程序。

* + 1. 实验调试及心得

执行程序结果如所图 5示。

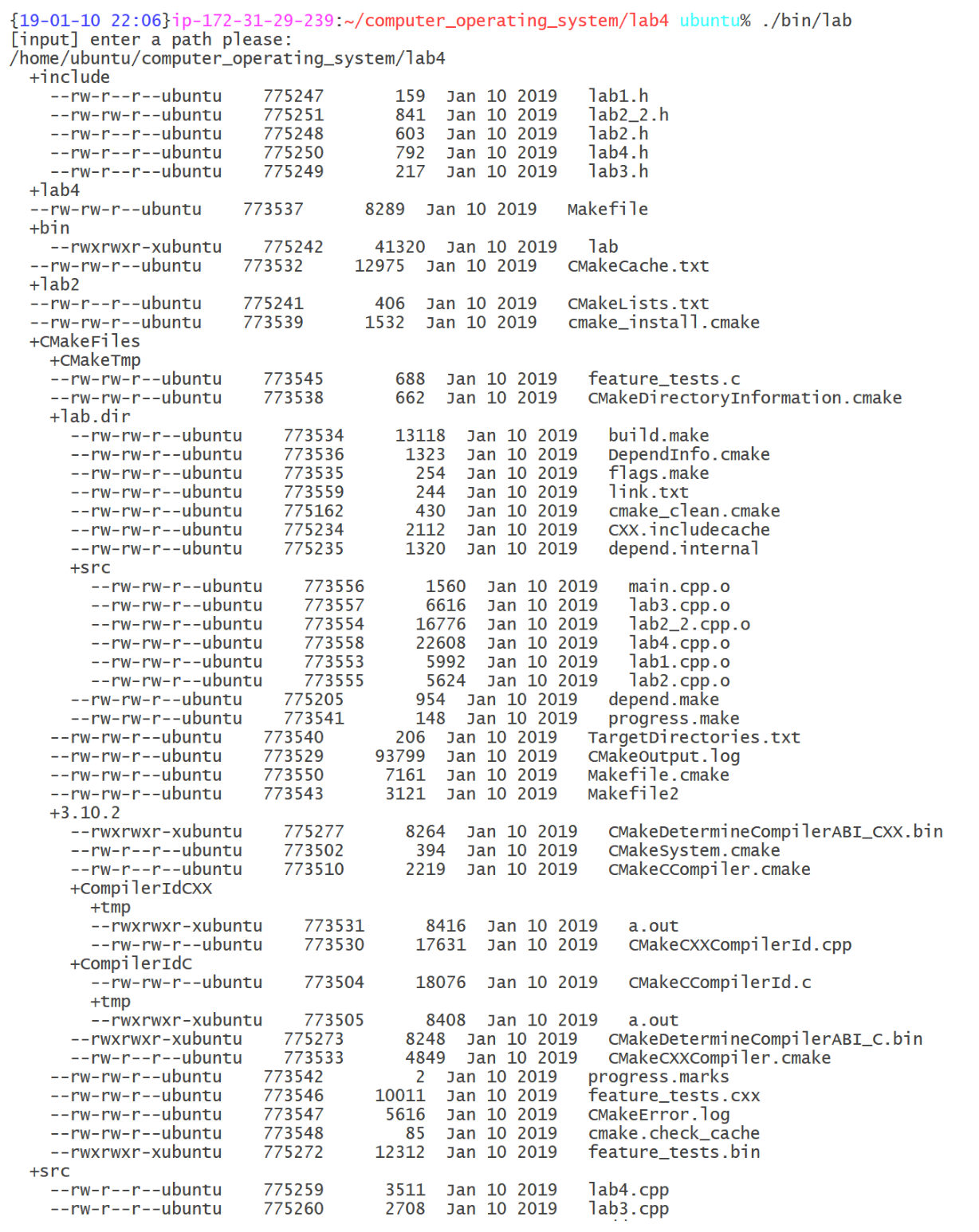


图 5 实验四执行结果

1. 附录 实验代码

## 5.1 实验一代码

* + 1. 头文件lab1.h

1. #ifndef LAB\_1\_H
2. #define LAB\_1\_H
4. **void** run\_lab1();
6. **int** child1();
7. **int** child2();
8. **void** killChild(**int** sig);
9. **void** recvSigInt(**int** signum);
10. #endif //!LAB\_1\_H%
    * 1. 源文件lab1.c
11. #include "../include/lab1.h"
13. #include<signal.h>
15. #include<unistd.h>
16. #include<iostream>
17. #include<cstring>
18. #include<sys/types.h>
19. #include<sys/wait.h>
21. pid\_t pid\_1;
22. pid\_t pid\_2;
23. **int** pipefd[2];
25. **void** run\_lab1()
26. {
27. pipe(pipefd);//create pip, pipefd[0] for read and pipefdp[0] for write
28. signal(SIGINT, recvSigInt);
29. signal(SIGALRM, recvSigInt);
30. alarm(10);
32. **if** ((pid\_1 = fork()) == 0) {
33. child1();
34. }
35. **if** ((pid\_2 = fork()) == 0) {
36. child2();
37. }
38. **int** status;
39. waitpid(pid\_1, &status, 0);
40. waitpid(pid\_2, &status, 0);
42. close(pipefd[0]);
43. close(pipefd[1]);
44. printf("Parent Process is Killed!");
45. **return**;
46. }
48. **int** child1(){
49. signal(SIGINT, SIG\_IGN);//屏蔽终止信号
50. signal(SIGUSR1, killChild);
51. **int** count = 0;
52. **char** str[30];
53. close(pipefd[0]);
54. **while**(**true**){
55. sprintf(str,"I send you %d times.", count++);
56. write(pipefd[1], str,20);
57. sleep(1);
58. }
59. }
60. **int** child2(){
61. signal(SIGINT,SIG\_IGN);//屏蔽终止信号
62. signal(SIGUSR2, killChild);
63. **char** str[30];
64. close(pipefd[1]);
65. **while**(**true**){
66. memset(str, '\0' ,30 );
67. read(pipefd[0], str, 20);
68. std::cout << str << std::endl;
69. }
70. }
72. **void** killChild(**int** sig){
73. **if**(sig == SIGUSR1){
74. std::cout << "Child Process l is Killed by Parent!" << std::endl;
75. }**else** **if**(sig == SIGUSR2){
76. std::cout << "Child Process 2 is Killed by Parent!" << std::endl;
77. }
78. close(pipefd[0]);
79. close(pipefd[1]);
81. exit(0);
82. }
84. **void** recvSigInt(**int** signum) {
85. kill(pid\_1, SIGUSR1);
86. kill(pid\_2, SIGUSR2);
87. }

## 5.2实验二代码

* + 1. 头文件lab2.cpp

1. #ifndef LAB\_2\_H
2. #define LAB\_2\_H
4. **void** run\_lab2();
5. **void** P(**int** semid, **int** index);
6. **void** V(**int** semid, **int** index);
7. **void** \*subp1(**void** \*arg);
8. **void** \*subp2(**void** \*arg);
10. **union** semun {
11. **int**              val;    /\* Value for SETVAL \*/
12. **struct** semid\_ds \*buf;    /\* Buffer for IPC\_STAT, IPC\_SET \*/
13. unsigned **short**  \*array;  /\* Array for GETALL, SETALL \*/
14. **struct** seminfo  \*\_\_buf;  /\* Buffer for IPC\_INFO
15. (Linux-specific) \*/
16. };
18. #define debug(message) std::cout<<message<<std::endl;
19. #endif //!LAB\_2\_H%
    * 1. 源文件lab2.cpp
20. #include"../include/lab2.h"
22. #include<sys/types.h>
23. #include<sys/ipc.h>
24. #include<sys/sem.h>
25. #include<iostream>
26. #include<unistd.h>
27. #include<pthread.h>
28. **int** semid;
30. **void** run\_lab2(){
31. key\_t semkey = 100;
32. semid = semget(semkey, 2 ,IPC\_CREAT | 0666);
33. **if**(semid < 0){
34. std::cout << "somthing wrong while semget" << std::endl;
35. exit(0);
36. }
37. **union** semun arg;
38. arg.val = 1;
39. **if**(semctl(semid, 0, SETVAL, arg)<0){
40. std::cout << "somthing wrong while set the semval" << std::endl;
41. exit(0);
42. }
43. arg.val = 0;
44. **if**(semctl(semid, 1, SETVAL, arg)<0){
45. std::cout << "somthing wrong while set the semval" << std::endl;
46. exit(0);
47. }
49. pthread\_t p1, p2;
50. **int** num = 0;
51. pthread\_create(&p1, NULL, subp1, &num);
52. pthread\_create(&p2, NULL, subp2, &num);
53. pthread\_join(p1, NULL);
54. pthread\_join(p2, NULL);
56. **if**(semctl(semid,  IPC\_RMID, 0)<0){
57. std::cout << "can't delete sem, somthing wrong"<< std::endl;
58. exit(0);
59. }
60. **return**;
61. }
63. **void** P(**int** semid, **int** index){
64. **struct** sembuf sem;
65. sem.sem\_num = index;
66. sem.sem\_op = -1;
67. sem.sem\_flg = 0;
68. semop(semid, &sem, 1);
69. **return**;
70. }
72. **void** V(**int** semid, **int** index){
73. **struct** sembuf sem;
74. sem.sem\_num = index;
75. sem.sem\_op = 1;
76. sem.sem\_flg = 0;
77. semop(semid, &sem, 1);
78. **return**;
79. }
81. **void** \*subp1(**void** \*arg){
82. **for**(**int** i = 1;i<=100;++i){
83. P(semid, 0);
84. (\*((**int** \*)(arg))) += i;

87. V(semid, 1);
88. //sleep(1);
89. }
90. }
92. **void** \*subp2(**void** \*arg){
93. **for**(**int** i = 1;i<=100;++i){
94. P(semid, 1);
95. std::cout << (\*((**int** \*)(arg))) << std::endl;
97. V(semid, 0);
98. //sleep(1);
99. }
100. }

## 5.3实验三代码

* + 1. 头文件header.h

1. #ifndef HEADER\_INCLUDE\_
2. #define HEADER\_INCLUDE\_
3. #include <pthread.h>
4. #include <sys/types.h>
5. #include <sys/sem.h>
6. #include <stdio.h>
7. #include <string.h>
8. #include <sys/ipc.h>
9. #include <sys/shm.h>
10. #include <unistd.h>
11. #include <sys/wait.h>
12. #include<stdlib.h>
13. **union** semum{
14. **int** val;
15. **struct** semid\_ds \*buf;
16. unsigned **short** \*array;
17. **struct** seminfo \*\_buf;
18. };
19. **void** P(**int** semid,**int** index)
20. {
21. **struct** sembuf sem;
22. sem.sem\_num=index;
23. sem.sem\_op=-1;
24. sem.sem\_flg=0;
25. semop(semid,&sem,1);
26. **return**;
27. }
28. **void** V(**int** semid,**int** index)
29. {
30. **struct** sembuf sem;
31. sem.sem\_num=index;
32. sem.sem\_op=1;
33. sem.sem\_flg=0;
34. semop(semid,&sem,1);
35. **return**;
36. }
37. **typedef** **struct** share\_buf1{
38. **char** buf[5][100];
39. **int** length[5];
40. }share\_buf1;
41. key\_t key1=1;
42. key\_t key2=100;
43. #endif //HEADER\_INCLUDE\_
    * 1. 源文件get.c
44. #include "header.h"
45. **int** main()
46. {
47. **int** seMID;
48. **int** semid1;
49. seMID=semget(key1,2,IPC\_CREAT|0666);
50. semid1=shmget(key2,**sizeof**(share\_buf1),IPC\_CREAT|0666);
51. share\_buf1 \*buf2=(share\_buf1 \*)shmat(semid1,NULL,0);
52. **FILE** \*fp;
53. **if**((fp=fopen("src","rb"))==NULL)
54. {
55. printf("open error");
56. }
57. **int** read\_num=0;
58. **int** size=0;
59. **while**(1)
60. {
61. P(seMID,0);
62. size=fread(buf2->buf[read\_num],**sizeof**(**char**),100,fp);
63. buf2->length[read\_num]=size;
64. read\_num=(read\_num+1)%5;
65. V(seMID,1);
66. **if**(size<100)
67. **break**;
68. }
69. fclose(fp);
70. }
    * 1. 源文件put.c
71. #include "header.h"
72. #include <sys/stat.h>
73. **int** main()
74. {
75. **int** seMID;
76. **int** semid1;
77. seMID=semget(key1,2,IPC\_CREAT|0666);
78. semid1=shmget(key2,**sizeof**(share\_buf1),IPC\_CREAT|0666);
79. share\_buf1 \*buf2=(share\_buf1 \*)shmat(semid1,NULL,0);
80. **FILE** \*fp1;
81. **if**((fp1=fopen("target","wb"))==NULL)
82. {
83. printf("open error");
84. }
85. **int** write\_num=0;
86. **int** size;
87. **while**(1)
88. {
89. P(seMID,1);
91. fwrite(buf2->buf[write\_num],**sizeof**(**char**),buf2->length[write\_num],fp1);
92. **if**(buf2->length[write\_num]<100){
93. **break**;
94. }
95. write\_num=(write\_num+1)%5;
96. V(seMID,0);
97. }
98. printf("write successful\n");
99. fclose(fp1);
100. **return** 0;
101. }
     * 1. 源文件main.c
102. #include "header.h"
103. **int** main()
104. {
105. **int** seMID;
106. **int** semid1;
108. seMID=semget(key1,2,IPC\_CREAT|0666);
110. semid1=shmget(key2,**sizeof**(share\_buf1),IPC\_CREAT|0666);
111. share\_buf1 \*buf2=(share\_buf1 \*)shmat(semid1,NULL,0);
112. **union** semum arg;
113. arg.val=5;
114. semctl(seMID,0,SETVAL,arg);
115. arg.val=0;
116. semctl(seMID,1,SETVAL,arg);
117. **int** p1,p2;
118. **if**((p1=fork())==0)
119. {
120. printf("start get...\n");
121. execl("./get","get",NULL);
122. }
123. **else** **if**((p2=fork())==0)
124. {
125. printf("start put...\n");
126. execl("./put","put",NULL);
127. }
128. **else**
129. {
130. waitpid(p1,NULL,0);
131. waitpid(p2,NULL,0);
132. semctl(seMID,1,IPC\_RMID,0);
133. shmctl(semid1,IPC\_RMID,0);
134. }
135. **return** 0;
136. }

## 5.4实验四代码

* + 1. 头文件lab4.h

1. #ifndef LAB\_4\_H
2. #define LAB\_4\_H
4. #include<sys/stat.h>
5. #include<string>

8. **class** ls{
9. std::string path;
11. **public**:
12. ls(std::string p){
13. **this**->path = p;
14. }
15. ~ls(){};
17. **void** ls\_l(){
18. ls\_l(**this**->path, 0);
19. }
20. **void** ls\_l(std::string path,**int** depth);
21. **private**:
22. //打印目录相关信息
23. **void** printDir(**const** **struct** dirent &dirst, **int** depth);
24. //打印文件相应信息
25. **void** printFile(**const** **struct** dirent & dirst, **const** **struct** stat &sb, **int** depth);
26. //生成权限描述符
27. std::string getMod(**const** **struct** stat & fileInfo);
28. std::string getUser(**const** **struct** stat & fileInfo);
29. std::string getInfoAndSizeAndTime(**const** **struct** stat & fileInfo, **const** **struct** dirent & dirst);
30. };
32. **void** run\_lab4();
33. #endif//!LAB\_4\_H%
    * 1. 源文件lab4.cpp
34. #include "../include/lab4.h"
35. #include<iostream>
36. #include<cstring>
37. #include<sys/types.h>
38. #include<sys/stat.h>
39. #include<dirent.h>//for DIR
40. #include<unistd.h>
41. #include<pwd.h>//for getpwuid


45. /\*
46. 目录打印样式
47. +dictionary name
48. -file name 1
49. -file name 2
50. +sub dictionary name
51. -file name 1
52. -file name 2
53. -file name 3
55. depth计算打印的空格数量，每层目录，空格加2
56. \*/
58. **void** ls::ls\_l(std::string path, **int** depth){
59. std::string pathBackup(path);
60. **struct** dirent \*entry = NULL;
61. **struct** stat fileInfo;
62. **struct** dirent\* dirst = NULL;
63. DIR \*dirp = opendir(path.c\_str());
64. **if**(dirp == NULL){
65. std::cout << "[error] open dir: " << path << " false" << std::endl;
66. exit(-1);
67. }
68. **for**(dirst = readdir(dirp); dirst; dirst = readdir(dirp)){
69. **if**(std::string(dirst->d\_name) == "." || (std::string(dirst->d\_name) == "..")){
70. **continue**;
71. }
72. lstat((path + "/" + dirst->d\_name).c\_str(), &fileInfo);
73. **if**(S\_ISDIR(fileInfo.st\_mode)){
74. printDir(\*dirst, depth);
75. ls\_l(path + "/" + dirst->d\_name, depth + 1);
76. }**else**{
77. printFile(\*dirst, fileInfo, depth);
78. }
79. }
80. }
82. **void** ls::printDir(**const** **struct** dirent &dirst, **int** depth){
83. **char** buffer[100];
84. memset(buffer, '\0', 100);
85. memset(buffer, ' ', (depth +1)\* 2);
86. std::string result(buffer);
87. result += "+";
88. result += dirst.d\_name;
89. std::cout << result << std::endl;
90. }
91. **void** ls::printFile(**const** **struct** dirent & dirst, **const** **struct** stat &fileInfo, **int** depth){
93. **char** buffer[100];
94. memset(buffer, '\0', 100);
95. memset(buffer, ' ', (depth +1)\* 2);
96. std::string result(buffer);
97. result += "-";
98. result = result + getMod(fileInfo) + getUser(fileInfo) + getInfoAndSizeAndTime(fileInfo, dirst);
99. std::cout << result << std::endl;
100. }
101. std::string ls::getMod(**const** **struct** stat & fileInfo){
102. mode\_t mode = fileInfo.st\_mode;
103. std::string formatedLine;
104. **return** formatedLine
105. + (S\_ISDIR(mode)?"d":"-")
106. + ((mode&S\_IRWXU)&S\_IRUSR?"r":"-")
107. + ((mode&S\_IRWXU)&S\_IWUSR?"w":"-")
108. + ((mode&S\_IRWXU)&S\_IXUSR?"x":"-")
109. + ((mode&S\_IRWXG)&S\_IRGRP?"r":"-")
110. + ((mode&S\_IRWXG)&S\_IWGRP?"w":"-")
111. + ((mode&S\_IRWXG)&S\_IXGRP?"x":"-")
112. + ((mode&S\_IRWXO)&S\_IROTH?"r":"-")
113. + ((mode&S\_IRWXO)&S\_IWOTH?"w":"-")
114. + ((mode&S\_IRWXO)&S\_IXOTH?"x":"-");
115. }
116. std::string ls::getUser(**const** **struct** stat & fileInfo){
117. **return** std::string(getpwuid(fileInfo.st\_uid)->pw\_name);
118. }
119. std::string ls::getInfoAndSizeAndTime(**const** **struct** stat & fileInfo, **const** **struct** dirent & dirst){
121. **char** info[1000];
122. **char** ctm[10];
123. **char** formatCtm[10];
124. memset(info, '\0', 100);
125. **struct** **tm** \* ltm;
126. ltm = localtime(&fileInfo.st\_mtime);
127. sprintf(info, "%10d%10d", fileInfo.st\_ino, fileInfo.st\_size);
128. std::string result(info);
129. strftime(ctm, 9, "%b", ltm);
130. sprintf(formatCtm, "%5s", ctm);
131. result += formatCtm;
132. strftime(ctm, 9, "%d", ltm);
133. sprintf(formatCtm, "%3s", ctm);
134. result += formatCtm;
135. strftime(ctm, 9, "%Y", ltm);
136. sprintf(formatCtm, "%5s", ctm);
137. result += formatCtm;
138. result += "   ";
139. result += dirst.d\_name;
140. **return** result;
141. }
143. **void** run\_lab4(){
144. std::cout << "[input] enter a path please:" << std::endl;
145. std::string path;
146. std::cin >> path;
147. ls l(path);
148. l.ls\_l();
149. **return**;
150. }%