



# 华中科技大学

## 操作系统原理实验报告

姓 名：龙际全  
学 院：计算机科学与技术  
专 业：计算机科学与技术  
班 级：CS1603  
学 号：U201614577  
指导教师：石柯

分数	
教师签名	

2018 年 01 月 04 日

# 目 录

<b>1 实验一 进程控制 .....</b>	<b>1</b>
1.1 实验目的.....	1
1.2 实验内容.....	1
1.3 实验设计.....	1
1.3.1 开发环境.....	1
1.3.2 实验设计.....	2
1.4 实验调试.....	2
1.4.1 实验步骤.....	2
1.4.2 实验调试及心得.....	3
附录 实验代码.....	4
lab1.c.....	4
<b>2 实验二 进程控制 .....</b>	<b>6</b>
2.1 实验目的.....	6
2.2 实验内容.....	6
2.3 实验设计.....	6
2.3.1 开发环境.....	6
2.3.2 实验设计.....	6
2.4 实验调试.....	7
2.4.1 实验步骤.....	7
2.4.2 实验调试及心得.....	7
附录 实验代码.....	8
lab2.c.....	8
<b>3 实验三 共享内存与进程同步.....</b>	<b>10</b>
3.1 实验目的.....	10
3.2 实验内容.....	10
3.3 实验设计.....	10
3.3.1 开发环境.....	10
3.3.2 实验设计.....	10
3.4 实验调试.....	11
3.4.1 实验步骤.....	11
3.4.2 实验调试及心得.....	12
附录 实验代码.....	13
lab3.c.....	13
readbuf.c.....	14
writebuf.c .....	15
<b>4 实验四 Linux 文件目录.....</b>	<b>17</b>
4.1 实验目的.....	17
4.2 实验内容.....	17

4.3 实验设计 .....	17
4.3.1 开发环境 .....	17
4.3.2 实验设计 .....	17
4.4 实验调试 .....	18
4.4.1 实验步骤 .....	18
4.4.2 实验调试及心得 .....	19
附录 实验代码 .....	20
walk_recursive.cpp .....	20
walk_bfs.cpp .....	22

# 1 实验一 进程控制

## 1.1 实验目的

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

## 1.2 实验内容

- (1) 编写程序,演示多进程并发执行和进程软中断、管道通信;
- (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);
- (3) 子进程 2 从管道读出信息,并显示在屏幕上;
- (4) 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 1 is Killed by Parent!  
Child Process 2 is Killed by Parent!
- (5) 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止:  
Parent Process is Killed!

## 1.3 实验设计

### 1.3.1 开发环境

- (1) 操作系统: Ubuntu 18.04.1 LTS;
- (2) 编译器: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0;
- (3) 编辑器: VIM - Vi IMproved 8.1。

### 1.3.2 实验设计

- (1) 数据结构
  - a) 两个 `pid_t` 型变量，用于创建进程的返回值；
  - b) 定义大小为 2 的 `int` 型数组，用作管道通信；
  - c) 一个初值为 1 的 `int` 型变量，用于发送次数计数；
  - d) 大小适度的 `char` 型数组，用于子进程 1 写数据和子进程 2 读数据。
- (2) 程序结构
  - a) 在 `main` 函数内创建管道，设置软中断信号 `SIGINT`，然后用 `fork` 创建进程。假如返回值为 0，则为子进程 1；若返回值大于 0，则为父进程，则在父进程中继续创建进程，如果返回值为 0，则为子进程 2。这个过程用 `if-else` 语句机构实现，各个条件下分别为父进程、子进程 1 和子进程 2 的执行代码。
  - b) 用一个中断处理函数 `myfunc` 来处理中断信号，在父进程、子进程 1 和子进程 2 中，分别用参数 `SIGINT`、`SIGUSR1` 和 `SIGUSR2` 调用该函数，执行的功能分别为：向两个子进程发送 `SIGUSR` 信号、退出子进程 1 和退出子进程 2。
  - c) 父进程中调用中断处理函数，当 `SIGINT` 信号到来时，用 `kill` 函数向子进程发送信号。等待子进程 1 和子进程 2 退出后，再关闭管道，退出自己。
  - d) 子进程 1 先忽略 `SIGINT` 信号，再调用中断处理函数，处理父进程发来的 `SIGUSR1` 信号，然后进入循环，向管道中写入数据，每写入一次休眠一秒。
- (3) 子进程 2 先忽略 `SIGINT` 信号，再调用中断处理函数，处理父进程发来的 `SIGUSR1` 信号，然后进入循环，从管道中读取数据，然后在标准输出打印，每读取一次也休眠一秒

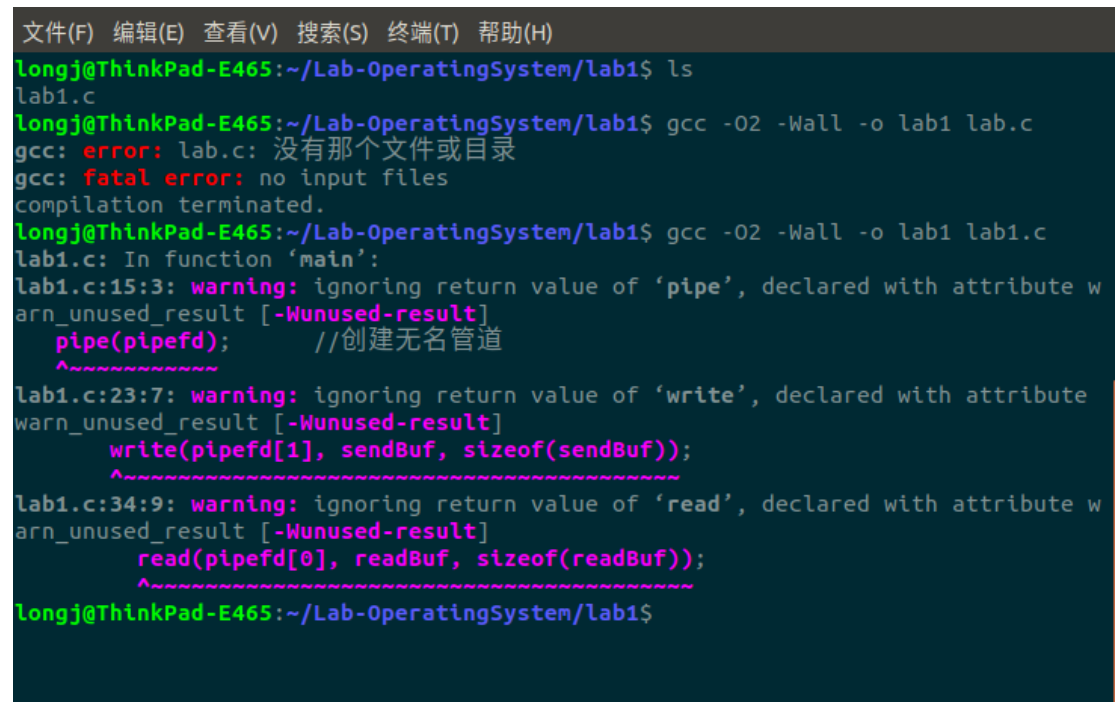
## 1.4 实验调试

### 1.4.1 实验步骤

- (1) 定义所有数据结构，把进程创建框架写好，暂不写实际执行代码，在每个进程创建后输出相关提示信息，使程序整体可以运行。
- (2) 完成进程功能代码和中断处理函数，再次编译运行，观察实验结果，与预期不符合就进行调试，直到结果正确。

## 1.4.2 实验调试及心得

- (1) 编译源程序 lab1.c, gcc -O2 -Wall -o lab1 lab1.c, 显示未使用函数返回值 Warning, 除此之外, 无任何 Warning 和 Error, 如图 1-1 编译无明显警告:



```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$ ls
lab1.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$ gcc -O2 -Wall -o lab1 lab.c
gcc: error: lab.c: 没有那个文件或目录
gcc: fatal error: no input files
compilation terminated.
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$ gcc -O2 -Wall -o lab1 lab1.c
lab1.c: In function 'main':
lab1.c:15:3: warning: ignoring return value of 'pipe', declared with attribute w
arn_unused_result [-Wunused-result]
    pipe(pipefd);           //创建无名管道
    ^~~~~~
lab1.c:23:7: warning: ignoring return value of 'write', declared with attribute
warn_unused_result [-Wunused-result]
    write(pipefd[1], sendBuf, sizeof(sendBuf));
    ^~~~~~
lab1.c:34:9: warning: ignoring return value of 'read', declared with attribute w
arn_unused_result [-Wunused-result]
    read(pipefd[0], readBuf, sizeof(readBuf));
    ^~~~~~
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$
```

图 1-1 编译无明显警告

- (2) 运行程序, 能够每秒输出一次, 而且序号是递增的; 按下 ctrl+c 后, 打印提示子进程 1 和子进程 2 依次结束, 然后父进程也正常退出, 程序结束, 如图 1-2 lab1 运行结果:

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lab1.c:23:7: warning: ignoring return value of 'write', declared with attribute
warn_unused_result [-Wunused-result]
    write(pipefd[1], sendBuf, sizeof(sendBuf));
    ^~~~~~
lab1.c:34:9: warning: ignoring return value of 'read', declared with attribute w
arn_unused_result [-Wunused-result]
    read(pipefd[0], readBuf, sizeof(readBuf));
    ^~~~~~
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$ ./lab1
I send you 1 times
I send you 2 times
I send you 3 times
I send you 4 times
I send you 5 times
I send you 6 times
I send you 7 times
I send you 8 times
I send you 9 times
I send you 10 times
I send you 11 times
^CChild process 2 is killed by parent
Child process 1 is killed by parent
parent process is killed!
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab1$
```

图 1-2 lab1 运行结果

- (3) 通过本次实验，进一步熟悉了 linux 系统的命令行操作，深入理解了进程的概念，知道了如何创建进程、让进程挂起、等待进程结束和杀死进程，实验难度不大，但实验过程中还是遇到了各种问题，主要是因为对于知识不熟悉，对于需要用到的各种函数和概念比较陌生。

## 附录 实验代码

### lab1.c

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
void HandleFunc(int sigNum);
pid_t pid1, pid2; //必须放到全局变量里，因为要给这两个子进程发送信号
int main(void) {
    int count = 1;    //发送次数
    int pipefd[2];    //用于两个子进程通信
    char sendBuf[100]; //实验中"i send you x times", 100 够用了吧
    char readBuf[100]; //读缓冲
    pipe(pipefd);      //创建无名管道
    pid1 = fork();
    if (pid1 == 0) { //子进程 1
        signal(SIGUSR1, HandleFunc);
```

```

    signal(SIGINT, SIG_IGN); //或者像下面这样写
    // signal(SIGINT,1);          //function=1 指忽略该类信号
    while (1) {
        sprintf(sendBuf, "I send you %d times\n", count);
        write(pipefd[1], sendBuf, sizeof(sendBuf));
        sleep(1);
        count++;
    }
} else {
    pid2 = fork();
    if (pid2 == 0) { //子进程 2
        signal(SIGUSR2, HandleFunc);
        signal(SIGINT, SIG_IGN); //或者像下面这样写
        // signal(SIGINT,1);          //function=1 指忽略该类信号
        while (1) {
            read(pipefd[0], readBuf, sizeof(readBuf));
            sleep(1);
            printf("%s", readBuf);
        }
    } else { //父进程
        signal(SIGINT, HandleFunc); //收到软中断信号，处理该信号
        wait(&pid1);
        wait(&pid2);
        close(pipefd[0]);
        close(pipefd[1]);
        printf("parent process is killed!\n");
        return 0;
    }
}
}
}
void HandleFunc(int sigNum) {
    if (SIGUSR1 == sigNum) {
        printf("Child process 1 is killed by parent\n");
        exit(0);
    }
    if (SIGUSR2 == sigNum) {
        printf("Child process 2 is killed by parent\n");
        exit(0);
    }
    if (SIGINT == sigNum) { //收到软中断信号，发送 sigusr1 和 sigusr2
        // signal(SIGUSR1, HandleFunc);
        // signal(SIGUSR2, HandleFunc);
        //函数写错了，signal 是用于信号处理的，发送信号应该用 kill
        kill(pid1, SIGUSR1);
        kill(pid2, SIGUSR2);
        // exit(0);          //还不能 exit 退出程序，因为还要回到 main 函数
        return;
    }
}

```



## 2 实验二 进程控制

### 2.1 实验目的

- (1) 掌握 Linux 下线程的概念;
- (2) 了解 Linux 线程同步与通信的主要机制;
- (3) 通过信号灯操作实现线程间的同步与互斥。

### 2.2 实验内容

设计并实现一个计算线程与一个 I/O 线程共享缓冲区的同步与通信, 要求:

- (1) 两个线程, 共享公共变量 a;
- (2) 线程 1 负责计算(1 到 100 的累加, 每次加一个数);
- (3) 线程 2 负责打印(输出累加的中间结果);
- (4) 主进程等待子线程退出。

### 2.3 实验设计

#### 2.3.1 开发环境

- (1) 操作系统: Ubuntu 18.04.1 LTS;
- (2) 编译器: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0;
- (3) 编辑器: VIM - Vi IMproved 8.1。

#### 2.3.2 实验设计

- (1) 数据结构
  - a) 整型变量 `semid`, 用于创建两个信号灯, 一个用于计算进程, 一个用于打印进程;
  - b) 整型变量 `sum`, 用于从 1 到 100 的求和;
  - c) 两个 `pthread_t` 线程变量, 用于创建运算线程和打印线程。
- (2) 程序结构
  - a) `main` 函数结构:
    - 创建信号灯并赋初值;

创建计算线程和打印线程；

等待两个线程运行结束；

删除信号灯并退出。

b) 计算线程 subp1 结构：

主体结构为 for 循环，从 1 开始计数，到 100 时循环结束。每次循环都先对信号灯 0 进行 P 操作，表示申请使用 sum 变量并对其求和。

假如打印线程当前没有使用 sum 变量，则计算线程可以改变 sum 的值，否则，线程挂起，等到打印线程释放 sum 时才可以执行。

每次计算完毕，对信号灯 1 进行 V 操作，激活打印线程，让对方知道有新数据可以打印。

c) 打印线程 subp2 结构：

主体结构也为 for 循环，循环 100 次。每次循环先对信号灯 1 进行 P 操作，申请读出 sum 的值，假如计算线程没有计算完毕，则打印线程还不能使用 sum 的值，将被挂起，直到计算线程对信号灯 1 进行 V 操作，才能继续打印。

打印线程成功读到 sum 的值以后，将其打印到标准输出，再对信号灯 0 进行 V 操作，保证计算线程可以进行下一轮计算。

## 2.4 实验调试

### 2.4.1 实验步骤

- (1) 根据方案设计，完成数据结构的定义和程序代码编写。
- (2) 编译程序，发现报错信息则修改代码，对于警告信息也尽可能想办法消除。
- (3) 生成可执行文件后运行程序，判断结果是否符合预期，不合要求的，继续返回上一步对代码进行修改。

### 2.4.2 实验调试及心得

- (1) 编译程序，`gcc -O2 -Wall -o lab2 lab2.c`，编译失败，如图 2-1 lab2 编译失败：

```

/tmp/ccqfKyp.o: 在函数‘main’中:
lab2.c:(.text.startup+0x5e): 对‘pthread_create’未定义的引用
lab2.c:(.text.startup+0x75): 对‘pthread_create’未定义的引用
lab2.c:(.text.startup+0x83): 对‘pthread_join’未定义的引用
lab2.c:(.text.startup+0x91): 对‘pthread_join’未定义的引用
collect2: error: ld returned 1 exit status
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$

```

图 2-1 lab2 编译失败

- (2) 上网看博客知道，Linux 线程库 pthread 并不是 gcc 所带的标准库，所以需要额外添加链接选项 -lpthread 链接进来，即 `gcc -O2 -Wall -o lab2 lab2.c -lpthread`，编译成功，无明显错误。
- (3) 运行程序，结果正确，如图 2-2 lab2 运行结果（只显示了最后 10 行）：

```

longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$ ls
lab2  lab2.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$ ./lab2 >> result.txt
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$ ls
lab2  lab2.c  result.txt
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$ tail -n 10 result.txt
thread 2 print:4656
thread 1 counting...
thread 2 print:4753
thread 1 counting...
thread 2 print:4851
thread 1 counting...
thread 2 print:4950
thread 1 counting...
thread 2 print:5050
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab2$

```

图 2-2 lab2 运行结果

- (4) 初次接触有关信号灯的操作，通过摸索了解了信号灯的工作机制，进一步理解了课堂中所学的知识，解决报错时学会了用 man 命令查看函数的头文件和函数功能。

## 附录 实验代码

### lab2.c

```

#include <linux/sem.h> //包含信号灯操作
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
void *compute_thread(); //计算线程
void *print_thread();   //打印线程

```

```

int semid; //信号量
int a = 0; //全局变量，供两个线程使用
pthread_t idOne, idTwo; //两个子线程的线程号
void P(int semid, int index);
void V(int semid, int index);
int main(void) {
    semid = semget(IPC_PRIVATE, 2, IPC_CREAT | 0666); //创建信号量
    semctl(semid, 0, SETVAL, 1);
    semctl(semid, 1, SETVAL, 0);
    pthread_create(&idOne, NULL, compute_thread, NULL);
    pthread_create(&idTwo, NULL, print_thread, NULL);
    pthread_join(idOne, NULL);
    pthread_join(idTwo, NULL);
    putchar('\n');
    semctl(semid, 0, IPC_RMID);
    return 0;
}
void *compute_thread() {
    int i;
    for (i = 1; i < 101; i++) {
        P(semid, 0);
        a = a + i;
        printf("thread 1 counting...\n");
        V(semid, 1);
    }
}
void *print_thread() {
    int i;
    for (i = 1; i < 101; i++) {
        P(semid, 1);
        printf("thread 2 print:%d\n", a);
        V(semid, 0);
    }
}
void P(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;
    sem.sem_flg = 0; //操作标记: 0 或 IPC_NOWAIT 等
    semop(semid, &sem, 1); // 1:表示执行命令的个数
    return;
}
void V(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;
    sem.sem_flg = 0;
    semop(semid, &sem, 1);
    return;
}

```

## 3 实验三 共享内存与进程同步

### 3.1 实验目的

- (1) 掌握 Linux 下共享内存的概念与使用方法;
- (2) 掌握环形缓冲的结构与使用方法;
- (3) 掌握 Linux 下进程同步与通信的主要机制。

### 3.2 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。

### 3.3 实验设计

#### 3.3.1 开发环境

- (1) 操作系统：Ubuntu 18.04.1 LTS;
- (2) 编译器：gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0;
- (3) 编辑器：VIM - Vi IMproved 8.1。

#### 3.3.2 实验设计

- (1) 数据结构
  - a) 整形数组 shmid，用于获取共享缓冲区;
  - b) 整形变量 semid，用于获取信号灯;
  - c) 字符指针数组 addr，用于存放缓冲区首地址;
  - d) 两个整形变量，用于创建子进程。
- (2) 程序结构
  - a) main 函数结构：  
创建包含 10 个 50 字节的共享内存组;  
创建两个信号灯，用于控制读写;  
对信号灯赋初值，分别是 10 和 0;  
等待两个子进程结束;

删除信号灯；  
删除共享内存组并退出。

- b) 写缓冲区进程 writebuf:
- 获取共享内存组；
  - 建立数组形式环形缓冲区；
  - 获取信号灯；
  - 打开源文件；
  - 循环：
    - 信号灯 P 操作；
    - 从文件读取数据存到缓冲区；
    - 移动环形缓冲区指针；
    - 信号灯 V 操作；
  - 上述循环当读到文件结束符时结束。
- c) 读缓冲区进程 readbuf:
- 获取共享内存组；
  - 建立数组形式环形缓冲；
  - 获取信号灯；
  - 创建文件；
  - 循环：
    - 信号灯 P 操作；
    - 从缓冲区读取数据；
    - 将数据写入文件；
    - 移动环形缓冲区指针；
    - 信号灯 V 操作；
  - 上述循环当读到结束标记时结束。

## 3.4 实验调试

### 3.4.1 实验步骤

- (1) 将实验方案设计中的主进程、子进程 1 和子进程 2 分成三个源文件，各子编译成可执行文件，在主进程的可执行文件中调用两个子进程的可执行文件。
- (2) 运行可执行程序，在文件夹中观察是否创建了新文件，若是，用 cmp 命令比较两个文件的内容是否相同。

(3) 假如实验结果不符合实验要求，则继续返回修改代码，直至达到要求。

### 3.4.2 实验调试及心得

(1) 编译主进程程序 lab3、读进程程序 readbuf、写进程程序 writebuf，均无明显错误，如图 3-1 lab3 三个程序编译成功：

```
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ls
lab3.c readbuf.c writebuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ gcc -g -o lab3 lab3.c
lab3.c: In function 'main':
lab3.c:37:5: warning: null argument where non-null required (argument 2) [-Wnonnull]
    execv("./readbuf", NULL);
    ^~~~~
lab3.c:40:5: warning: null argument where non-null required (argument 2) [-Wnonnull]
    execv("./writebuf", NULL);
    ^~~~~
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ gcc -g -o readbuf readbuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ gcc -g -o writebuf writebuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ls
lab3 lab3.c readbuf readbuf.c writebuf writebuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$
```

图 3-1 lab3 三个程序编译成功

(2) 运行程序 lab3，输入文件为 input，输出文件为 output；  
(3) 比较两个文件 input 和 output 是否有差异，cmp input output，如图 3-2 文件誊抄成功，没有输出表示没有差异；

```
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ls
lab3 lab3.c readbuf readbuf.c writebuf writebuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ cp ~/图片/wallpaper.jpg ./input
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ls
input lab3 lab3.c readbuf readbuf.c writebuf writebuf.c
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ./lab3 >> debug.log
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ls
debug.log lab3 output readbuf.c writebuf.c
input lab3.c readbuf writebuf
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ cmp input output
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$
```

图 3-2 文件誊抄成功

(4) 运行 ipcs -m 命令，查看共享内存，如下：

```
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ ipcs -m >> result.txt
longj@ThinkPad-E465:~/Lab-OperatingSystem/lab3$ tail -n 5 result.txt
键          semid      拥有者  权限      nsems
0x510a002c  32768      longj   666       1
0x510a003e  98305      longj   666       1
0x510a004f  196610     longj   666       1
```

图 3-3 查看 lab3 共享内存

(5) 这次实验相当于前两次实验的综合，进一步熟悉了信号灯，学会了共享缓

冲区的使用，也学会了相较于 fopen 函数更为底层的 open 函数读取文件的操作。

## 附录 实验代码

### lab3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>
#define SHMKEY 7777
#define SEMKEY 9999
int i = 0;
int shmid[20];
int sid = 0;
union semunarg {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
    struct seminfo *_buf;
} arg;
int main() {
    sid = semget((key_t)SEMKEY, 2, IPC_CREAT | 0666);
    if (sid == -1)
        return -1;
    arg.val = 0;
    semctl(sid, 0, SETVAL, arg);
    arg.val = 1;
    semctl(sid, 1, SETVAL, arg);
    for (i = 0; i < 10; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), 1024, 0666 | IPC_CREAT);
    }
    for (i = 10; i < 20; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), sizeof(int), 0666 | IPC_CREAT);
    }
    int p1, p2;
    if ((p1 = fork()) == 0) {
        puts("Create readbuf!\n");
        execv("./readbuf", NULL);
    } else if ((p2 = fork()) == 0) {
        puts("Create writebuf!\n");
        execv("./writebuf", NULL);
    }
    wait(0);
    wait(0);
    semctl(sid, 0, IPC_RMID);
    for (i = 0; i < 20; i++) {
        shmctl(shmid[i], IPC_RMID, 0);
    }
}
```



```

    }
    return 0;
}

```

## readbuf.c

```

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define SHMKEY 7777
#define SEMKEY 9999
int i = 0;
int shmid[20];
int sid = 0;
char *addr[10];
int *len[10];
void P(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;
    sem.sem_flg = 0;
    semop(semid, &sem, 1);
    return;
}
void V(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;
    sem.sem_flg = 0;
    semop(semid, &sem, 1);
    return;
}
void readbuf() {
    int fp = open("./input", O_RDONLY);
    P(sid, 1);
    puts("read begin");
    for (i = 0; (*(len[i]) = read(fp, addr[i], 1024)) != 0;) {
        puts("read!");
        V(sid, 0);
        if (i == 9) {
            P(sid, 1);
            i = 0;
        } else
            i++;
    }
    puts("read over!");
    close(fp);
    return;
}

```

```

int main() {
    puts("readbuf begin!");
    for (i = 0; i < 10; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), 1024, 0666);
        addr[i] = shmat(shmid[i], 0, 0);
    }
    for (i = 10; i < 20; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), sizeof(int), 0666);
        len[i - 10] = shmat(shmid[i], 0, 0);
    }
    sid = semget((key_t)SEMKEY, 2, 0666);
    readbuf();
    for (i = 0; i < 20; i++) {
        shmdt(addr[i]);
    }
    return 0;
}

```

## writebuf.c

```

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define SHMKEY 7777
#define SEMKEY 9999
int i = 0;
int shmid[20];
int sid = 0;
char *addr[10];
int *len[10];
void P(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;
    sem.sem_flg = 0;
    semop(semid, &sem, 1);
    return;
}
void V(int semid, int index) {
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;
    sem.sem_flg = 0;
    semop(semid, &sem, 1);
    return;
}
void writebuf() {
    int fp = open("./output", O_WRONLY | O_CREAT);
    for (i = 0; *(len[i]) != 0;) {

```

```

    P(sid, 0);
    write(fp, addr[i], *len[i]);
    puts("write!");
    if (i == 9) {
        i = 0;
        V(sid, 1);
    } else
        i++;
}
puts("write over!");
close(fp);
return;
}
int main() {
    puts("writebuf begin!");
    for (i = 0; i < 10; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), 1024, 0666);
        addr[i] = shmat(shmid[i], 0, 0);
    }
    for (i = 10; i < 20; i++) {
        shmid[i] = shmget((key_t)(SHMKEY + i), sizeof(int), 0666);
        len[i - 10] = shmat(shmid[i], 0, 0);
    }
    sid = semget((key_t)SEMKEY, 2, 0666);
    writebuf();
    for (i = 0; i < 20; i++) {
        shmdt(addr[i]);
    }
    return 0;
}

```

## 4 实验四 Linux 文件目录

### 4.1 实验目的

- (1) 了解 Linux 文件系统与目录操作;
- (2) 了解 Linux 文件系统目录结构;
- (3) 掌握文件和目录的程序设计方法。

### 4.2 实验内容

编程实现目录查询功能:

- (1) 功能类似 `ls -lR`;
- (2) 查询指定目录下的文件及子目录信息;
- (3) 显示文件的类型、大小、时间等信息;
- (4) 递归显示子目录中的所有文件信息;
- (5) 改进程序, 使用非递归方式显示目录中的所有文件信息。

### 4.3 实验设计

#### 4.3.1 开发环境

- (1) 操作系统: Ubuntu 18.04.1 LTS;
- (2) 编译器: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0;
- (3) 编辑器: VIM - Vi Improved 8.1。

#### 4.3.2 实验设计

- (1) 将实验 PPT 所给框架翻译为 C 语言代码;
- (2) 采用深度递归遍历 Linux 文件系统, 使用相对路径作为工作路径;
- (3) 非递归方式下, 改用绝对路径配合广度优先遍历 BFS。

## 4.4 实验调试

### 4.4.1 实验步骤

(1) 深度递归遍历文件数小的目录，正常运行，如下图 4-1:

```
[longj@ThinkPad-E465 lab4]$ pwd
/home/longj/Lab-OperatingSystem/lab4
[longj@ThinkPad-E465 lab4]$ ls
file_info.txt  walk_bfs.cpp  walk_recursive.cpp  walk_recursive.out
[longj@ThinkPad-E465 lab4]$ ./walk_recursive.out
--w-r--r-- 1 longj longj 3769 2019-01-08 22:52 walk_bfs.cpp 0
--w-r--r-- 1 longj longj 10630 2019-01-07 23:00 file_info.txt 0
--wxr-xr-x 1 longj longj 18736 2019-01-10 23:13 walk_recursive.out 0
--w-r--r-- 1 longj longj 2528 2019-01-10 23:13 walk_recursive.cpp 0
4
[longj@ThinkPad-E465 lab4]$
```

图 4-1 深度递归遍历小文件数目录

(2) 在根目录下做深度递归压力测试，正常运行，如下图 4-2:

```
[longj@ThinkPad-E465 /]$ pwd
/
[longj@ThinkPad-E465 /]$ ls
bin      home      media    run      tmp      vmlinuz
boot     lib       mnt      sbin     usr      vmlinuz.old
cdrom    lib32     opt      snap     var      walk_bfs.out
dev      lib64     proc     srv      initrd.img  walk_recursive.out
etc      lost+found root     sys      initrd.img.old
[longj@ThinkPad-E465 /]$ sudo ./walk_recursive.out > ~/messtest/file_info.txt
[longj@ThinkPad-E465 /]$ tail -6 ~/messtest/file_info.txt
--wxr-xr-x 1 root root 157224 2017-12-02 17:27 cpio 4
--wxr-xr-x 1 root root 26728 2017-12-01 03:38 ntfsctat 4
--wxr-xr-x 1 root root 30904 2018-01-18 17:43 true 4
--wxrwxrwx 1 root root 6 2018-08-17 15:15 bzegrep 4
--wxrwxrwx 1 root root 33 2019-01-08 09:47 initrd.img 0
1547050
[longj@ThinkPad-E465 /]$
```

图 4-2 深度递归遍历根目录

(3) 广度非递归遍历文件数小的目录，正常运行，如下图 4-3:

```

[longj@ThinkPad-E465 lab4]$ pwd
/home/longj/Lab-OperatingSystem/lab4
[longj@ThinkPad-E465 lab4]$ ls
file_info.txt  walk_bfs.out      walk_recursive.out
walk_bfs.cpp   walk_recursive.cpp
[longj@ThinkPad-E465 lab4]$ ./walk_bfs.out
--wxr-xr-x 1 longj longj 19408 2019-01-10 23:24 walk_bfs.out  4
--w-r--r-- 1 longj longj 3769 2019-01-08 22:52 walk_bfs.cpp  4
--w-r--r-- 1 longj longj 10630 2019-01-07 23:00 file_info.txt 4
--wxr-xr-x 1 longj longj 18736 2019-01-10 23:13 walk_recursive.out 4
--w-r--r-- 1 longj longj 2528 2019-01-10 23:13 walk_recursive.cpp 4
directory /home/longj/Lab-OperatingSystem/lab4 has 5 file in total

file num total : 5

```

图 4-3 广度非递归遍历小文件数目录

(4) 在根目录下做广度非递归压力测试，正常运行，如下图 4-4:

```

[longj@ThinkPad-E465 /]$ pwd
/
[longj@ThinkPad-E465 /]$ ls
bin      home      media    run      tmp      vmlinuz
boot     lib       mnt      sbin     usr      vmlinuz.old
cdrom    lib32     opt      snap     var      walk_bfs.out
dev      lib64     proc     srv      initrd.img  walk_recursive.out
etc      lost+found root     sys      initrd.img.old
[longj@ThinkPad-E465 /]$ sudo ./walk_bfs.out > ~/桌面/file_info.txt
[longj@ThinkPad-E465 /]$ tail -6 ~/桌面/file_info.txt
--w-rw-r-- 1 longj longj 1661 2017-11-28 21:06 sax.py 104
--w-rw-r-- 1 longj longj 0 2017-11-28 21:06 __init__.py 104
directory //home/longj/.local/share/Trash/files/Qt5.10.0/5.10.0/Src/qtwebengine/
src/3rdparty/chromium/third_party/WebKit/Tools/Scripts/webkitpy/thirdparty/wpt/w
pt/tools/html5lib/html5lib/treeadapters has 2 file in total

file num total : 1593832
[longj@ThinkPad-E465 /]$

```

图 4-4 广度非递归遍历根目录

## 4.4.2 实验调试及心得

- (1) 本次实验加深了我对 Linux 文件系统的理解，通过本次实验的锻炼，我已经能熟练编写程序了解 Linux 文件目录结构；
- (2) 本次实验锻炼了我解决问题的能力，第一次编写的深度递归遍历程序在根目录下程序栈溢出，于是通过更细致的查阅 Linux 文件 API 文档，我将程序改为了使用广度非递归遍历，减少了不必要的函数调用，使得程序更加健壮，容错性更强；
- (3) 通过查阅操作系统相关知识，了解到一个典型的进程应该包括代码区、栈区、堆区、静态区、全局区等内存区，其中局部变量在栈上，动态分配的内存存在

堆上，因此我想到可以将深度递归遍历时的局部变量改为动态分配内存，递归调用时只有参数压栈以此减轻栈的负担，Linux 默认栈大小 8M，文件深度不超过  $10^5$  量级（现实中这种情况也不存在）时，足以支撑程序完整的递归调用，果然，程序改进之后在根目录做压力测试后也没有出现栈溢出异常。

## 附录 实验代码

### walk\_recursive.cpp

```
#include <dirent.h>
#include <grp.h>
#include <iostream>
#include <pwd.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
using namespace std;
int num;
void printfile(int &depth);
void printinfo(struct stat &sb, char *name);
int main() {
    int depth = 0;
    printfile(depth);
    cout << num << endl;
    return 0;
}
void printfile(int &depth) {
    DIR *d; //声明一个句柄
    struct dirent *file; // readdir 函数的返回值就存放在这个结构体中
    struct stat sb;
    if (!(d = opendir("."))) {
        printf("error opendir\n");
        return;
    }
    while ((file = readdir(d)) != NULL) {
        if (strcmp(file->d_name, ".") == 0 || strcmp(file->d_name, "..") == 0)
            continue;
        if (lstat(file->d_name, &sb) < 0) {
            continue;
        }
        num++;
        if (S_ISDIR(sb.st_mode)) {
            printinfo(sb, file->d_name);
            cout << " " << depth << endl;
            depth += 4;
            string s = "/";
            s += file->d_name;
            if (chdir(s.c_str()) == 0)
```

```

        printf(depth);
    } else {
        printf(sb, file->d_name);
        cout << " " << depth << endl;
    }
}
closedir(d);
if (depth != 0)
    chdir("../");
depth -= 4;
}
void printinfo(struct stat &sb, char *name) {
    struct passwd *pd;
    struct group *gp;
    struct tm t;
    tzset();
    localtime_r(&(sb.st_mtime), &t);
    char buf[30];
    strftime(buf, 30, "%Y-%m-%d %H:%M", &t);

    if (S_ISDIR(sb.st_mode))
        cout << 'd';
    else
        cout << '-';
    if ((S_IRUSR & sb.st_mode) == S_IRGRP)
        cout << 'r';
    else
        cout << '-';
    if ((S_IWUSR & sb.st_mode) == S_IWUSR)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXUSR & sb.st_mode) == S_IXUSR)
        cout << 'x';
    else
        cout << '-';
    if ((S_IRGRP & sb.st_mode) == S_IRGRP)
        cout << 'r';
    else
        cout << '-';
    if ((S_IWGRP & sb.st_mode) == S_IWGRP)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXGRP & sb.st_mode) == S_IXGRP)
        cout << 'x';
    else
        cout << '-';
    if ((S_IROTH & sb.st_mode) == S_IROTH)
        cout << 'r';
    else
        cout << '-';
    if ((S_IWOTH & sb.st_mode) == S_IWOTH)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXOTH & sb.st_mode) == S_IXOTH)

```



```

        cout << 'x';
    else
        cout << '-';
    cout << " ";
    cout << sb.st_nlink;
    cout << " ";
    pd = getpwuid(sb.st_uid);
    if (pd != NULL)
        cout << pd->pw_name << " ";
    gp = getgrgid(sb.st_gid);
    if (gp != NULL)
        cout << gp->gr_name << " ";
    cout << sb.st_size << " ";
    cout << buf << " ";
    cout << name << " ";
}

```

## walk\_bfs.cpp

```

#include <dirent.h>
#include <grp.h>
#include <iostream>
#include <pwd.h>
#include <queue>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
using namespace std;
int num;
void printfile_bfs(int &depth);
void printinfo(struct stat &sb, char *name);
int main(int argc, char *argv[]) {
    int depth = 0;
    printfile_bfs(depth);
    printf("\nfile num total : %d\n", num);
    return 0;
}

void printfile_bfs(int &depth) {
    queue<char *> dir_queue;
    dir_queue.push(getcwd(NULL, PATH_MAX));
    while (dir_queue.empty() == false) {
        int this_level_count = dir_queue.size();
        depth += 4; // depth increase only when walk to next level
        for (int i = 0; i < this_level_count; ++i) {
            char *cur_dir_str = dir_queue.front();
            dir_queue.pop();
            DIR *dir_ptr = opendir(cur_dir_str);
            chdir(cur_dir_str);
            struct dirent *file_ptr;
            struct stat file_stat;
            if (dir_ptr == NULL) {
                printf("error open directory %s\n", cur_dir_str);
            }
        }
    }
}

```

```

        continue;
    }
    int file_count = 0;
    while ((file_ptr = readdir(dir_ptr)) != NULL) {
        if (lstat(file_ptr->d_name, &file_stat) < 0) {
            printf("error stat file %s info \n", file_ptr->d_name);
            continue;
        }
        if ((strcmp(file_ptr->d_name, ".") == 0) ||
            (strcmp(file_ptr->d_name, "..") == 0)) {
            // current directory or father directory
            continue;
        }
        num++;
        file_count++;
        if (S_ISDIR(file_stat.st_mode)) { // the file is directory
            printinfo(file_stat, file_ptr->d_name);
            cout << " " << depth << endl;
            // char *child_dir = (char *)malloc(sizeof(char) * PATH_MAX);
            // strcpy(child_dir, cur_dir_str);
            // strcat(child_dir, "/");
            // strcat(child_dir, file_ptr->d_name);
            // dir_queue.push(child_dir);
            string child_dir = cur_dir_str;
            child_dir += "/";
            child_dir += file_ptr->d_name;
            char *child_dir_name =
                (char *)malloc(sizeof(char) * (child_dir.size() + 1));
            strcpy(child_dir_name, child_dir.c_str());
            dir_queue.push(child_dir_name);
        } else {
            printinfo(file_stat, file_ptr->d_name);
            cout << " " << depth << endl;
        }
    }
    printf("directory %s has %d file in total\n\n", cur_dir_str, file_count);
    closedir(dir_ptr);
}
}
}

```

```

void printinfo(struct stat &sb, char *name) {
    struct passwd *pd;
    struct group *gp;
    struct tm t;
    tzset();
    localtime_r(&(sb.st_mtime), &t);
    char buf[30];
    strftime(buf, 30, "%Y-%m-%d %H:%M", &t);

    if (S_ISDIR(sb.st_mode))
        cout << 'd';
    else
        cout << '-';
    if ((S_IRUSR & sb.st_mode) == S_IRGRP)
        cout << 'r';
    else

```

```

        cout << '-';
    if ((S_IWUSR & sb.st_mode) == S_IWUSR)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXUSR & sb.st_mode) == S_IXUSR)
        cout << 'x';
    else
        cout << '-';
    if ((S_IRGRP & sb.st_mode) == S_IRGRP)
        cout << 'r';
    else
        cout << '-';
    if ((S_IWGRP & sb.st_mode) == S_IWGRP)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXGRP & sb.st_mode) == S_IXGRP)
        cout << 'x';
    else
        cout << '-';
    if ((S_IROTH & sb.st_mode) == S_IROTH)
        cout << 'r';
    else
        cout << '-';
    if ((S_IWOTH & sb.st_mode) == S_IWOTH)
        cout << 'w';
    else
        cout << '-';
    if ((S_IXOTH & sb.st_mode) == S_IXOTH)
        cout << 'x';
    else
        cout << '-';
    cout << " ";
    cout << sb.st_nlink;
    cout << " ";
    pd = getpwuid(sb.st_uid);
    if (pd != NULL)
        cout << pd->pw_name << " ";
    gp = getgrgid(sb.st_gid);
    if (gp != NULL)
        cout << gp->gr_name << " ";
    cout << sb.st_size << " ";
    cout << buf << " ";
    cout << name << " ";
}

```