華中科技大學

课 程 实 验 报 告

课程名称： 操 作 系 统 原 理

专业班级： **cs1609 班**

学　　号： **U201614749**

姓　　名： 宋君平

指导教师： 谢夏

报告日期： **2019.1.5**

计算机科学与技术学院

目录

实验一：进程控制

一、实验目的

1、掌握 Linux 系统用户界面中键盘命令的使用。

2、学会一种 Linux 下的编程环境。

3、掌握 Linux 下进（线）程的概念。

4、了解 Linux 进程同步与通信的主要机制，并通过信号灯操作实现进程间的同步与互斥。

二、实验内容

1、程序要求

两个线程,共享公共变量 a

线程 1 负责计算(+1)

线程 2 负责打印

2、运行环境

软件配置（含操作系统版本）：Ubuntu 14.1.2 build-8497320

VMware® Workstation 14 Pro

硬件：内存 2GB

处理器 1

硬盘（SCSI） 20GB

网络适配器 NAT

任意监视器最大分辨率 2560*160**0**

3、源程序

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<signal.h>
#include<stdlib.h>
#include<unistd.h>
#define MAXBUFFER 102
static pid_t pid1; //子进程 1
static pid_t pid2; //子进程 2
void sigmain(int sig)  //sigmain
{
```

```
    kill(pid1,SIGUSR1);
    kill(pid2,SIGUSR1);
}
void sigpid1(int sig) //sigpid1
{
    printf("Child Process 1 is Killed by Parent!\n");
    exit(0);
}
void sigpid2(int sig)
{
    printf("Child Process 2 is Killed by Parent!\n");
    exit(0);
}

int main()
{
    int status;
    int pipefd[2];
    //int result=pipe(pipefd);
        pipe(pipefd);
    //pipefd[0]只能用于读;  pipe[1]只能用于写

    char buffer[MAXBUFFER]; //缓冲区
    char buffer2[MAXBUFFER];
    pid1=fork();
     if(pid1==0)
     {
        signal(SIGINT,SIG_IGN);  //设置忽略信号量
        signal(SIGUSR1,sigpid1);
        close(pipefd[0]); //写入时关闭读通道
        int count=1;
        while(1)
        {
            sprintf(buffer,"I send you %d times.\n",count);
            //lockf(pipefd[1],1,0);
            write(pipefd[1],buffer,sizeof(buffer));
            count++;
            //lockf(pipefd[1],0,0);
            sleep(1);
         }
     }
    /*if(pid2==-1)
    {
        printf("failed to fork pid2.\n");
```

```
        return -1;
    }*/
    else
{
    pid2=fork();
    if(pid2==0)
    {
        signal(SIGINT,SIG_IGN);  //设置忽略信号量
        signal(SIGUSR1,sigpid2);
        //close(pipefd[1]); //读入时关闭写通道
        while(1)
        {
            read(pipefd[0],buffer,sizeof(buffer)); //读入管道
            //buffer2[size]='\0';
            //lockf(pipefd[0],0,0);
            printf("%s",buffer);
                        memset(buffer,'\0',MAXBUFFER);
         }
    }
    else
{
    signal(SIGINT,sigmain);
    waitpid(pid1,&status,0);
    waitpid(pid2,&status,0);
    close(pipefd[0]);//关闭读管道
    close(pipefd[1]);//关闭写管道
    printf("Parent Process is Killed!\n");
    exit(0);
}
}
} #include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<signal.h>
#include<stdlib.h>
#include<unistd.h>
#define MAXBUFFER 102


static pid_t pid1; //子进程1
static pid_t pid2; //子进程2

void sigmain(int sig)  //sigmain
```

```
{
    kill(pid1,SIGUSR1);
    kill(pid2,SIGUSR1);
}

void sigpid1(int sig) //sigpid1
{
    printf("Child Process 1 is Killed by Parent!\n");
    exit(0);
}

void sigpid2(int sig)
{
    printf("Child Process 2 is Killed by Parent!\n");
    exit(0);
}

int main()
{
    int status;
    int pipefd[2];
    //int result=pipe(pipefd);
        pipe(pipefd);
    //pipefd[0]只能用于读;  pipe[1]只能用于写

    char buffer[MAXBUFFER]; //缓冲区
    char buffer2[MAXBUFFER];
    pid1=fork();
     if(pid1==0)
     {
        signal(SIGINT,SIG_IGN);  //设置忽略信号量
        signal(SIGUSR1,sigpid1);
        close(pipefd[0]); //写入时关闭读通道
        int count=1;
        while(1)
        {
            sprintf(buffer,"I send you %d times.\n",count);
            //lockf(pipefd[1],1,0);
            write(pipefd[1],buffer,sizeof(buffer));
            count++;
            //lockf(pipefd[1],0,0);
            sleep(1);
         }
     }
```

```
    /*if(pid2==-1)
    {
        printf("failed to fork pid2.\n");
        return -1;
    }*/
    else
{
    pid2=fork();
    if(pid2==0)
    {
        signal(SIGINT,SIG_IGN);  //设置忽略信号量
        signal(SIGUSR1,sigpid2);
        //close(pipefd[1]); //读入时关闭写通道
        while(1)
        {
            read(pipefd[0],buffer,sizeof(buffer)); //读入管道
            //buffer2[size]='\0';
            //lockf(pipefd[0],0,0);
            printf("%s",buffer);
                        memset(buffer,'\0',MAXBUFFER);
         }
    }
    else
{
    signal(SIGINT,sigmain);
    waitpid(pid1,&status,0);
    waitpid(pid2,&status,0);
    close(pipefd[0]);//关闭读管道
    close(pipefd[1]);//关闭写管道
    printf("Parent Process is Killed!\n");
    exit(0);
}
}
}
```

4、实验结果

编写完成后测试实验结果如下图所示：

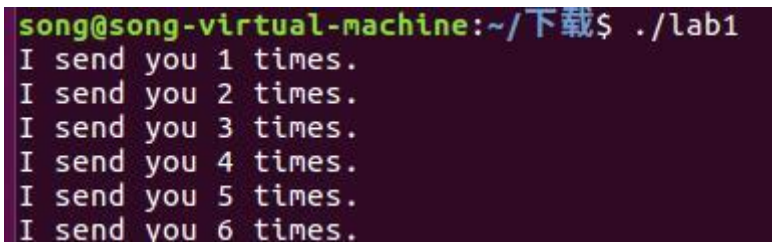

图 1-1.1 实验结果截图

结果中可以看到每过 1 秒控制台就会显示内容，而此时键盘的输入也不会影响输出结果：

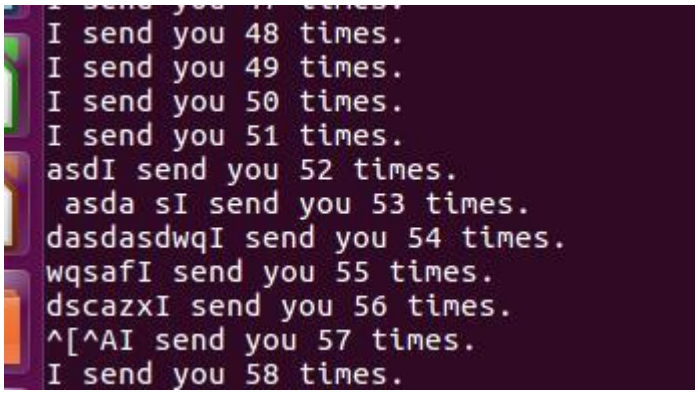

图 2-1.1

由此可知，实验结果正确。

三、实验心得

通过本次实验加深了我对于进程的理解。进一步认识了并发的实质。也学习到了通过未命名管道对进程互斥的控制方法。对于进程的控制有了进一步的理解。对进程的三态------就绪态，等待态，运行态有了更进一步的理解。处于等待态的进程不能直接变换到运行态，需要等待资源的调度，当进程处于就绪态时，进程只需要获得 cpu 的使用权即可进入运行态。运行态的进程运行之后转入等待态。

此次实验中，对于进程创建也有了进一步理解，子进程的进程号为 0 父进程大于 0 创建失败时为-1。需要注意的是创建多个进程时不要连续创建，需要在条件语句中判断，否则会造成创建更多个进程的情况，导致程序无法控制。

此次实验中还学到很多进程控制函数，进程修改函数，进程退出函数，进程等待函数。为下一次实验打下了基础。

实验二：线程同步与通信

一、实验目的

1、掌握 Linux 系统用户界面中键盘命令的使用。

2、学会一种 Linux 下的编程环境。

3、掌握 Linux 下进（线）程的概念。

4、了解 Linux 进程同步与通信的主要机制，并通过信号灯操作实现进程间的同步与互斥。

二、实验内容

1、程序要求

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)

2、运行环境

软件配置（含操作系统版本）：Ubuntu 14.1.2 build-8497320

VMware® Workstation 14 Pro

硬件：内存 2GB

处理器 1

硬盘（SCSI） 20GB

网络适配器 NAT

任意监视器最大分辨率 2560*1600

3、源程序

```
#include<pthread.h>
#include<sys/types.h>
#include<sys/sem.h>
#include<stdio.h>
#include<stdlib.h>
#define KEY 1 //信号量初始值
union semun{
   int val;
   struct semid_ds *buf;
   unsigned short *array;
   struct seminfo *_buf;

};

pthread_t p1,p2; //线程
int mutex;
static int flag=0;
int i=1;
```

```
int a=0;//计算和
void *subp1(); //计算线程
void *subp2(); //打印线程

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;
}


int main()
{

    mutex=semget(IPC_PRIVATE,2,IPC_CREAT | 0666);

    semctl(mutex,0,SETVAL,0);

    semctl(mutex,1,SETVAL,1);

    pthread_create(&p1,NULL,subp1,NULL);
    pthread_create(&p2,NULL,subp2,NULL);
pthread_join(p1,NULL);
    pthread_join(p2,NULL);
    if(semctl(mutex,0,IPC_RMID,0)==-1)
       {
           printf("信号灯删除失败，异常返回\n");
              exit(1);
       }
        printf("信号灯删除成功，正常返回\n");
```

```
    return 0;

}

void *subp1()
{
     for(;i<=100;i++){
        P(mutex,1);
        a=a+i;
        V(mutex,0);
    }
    flag=1;
 return NULL;
}

void *subp2()
{
    while(1)
    {
       P(mutex,0);
       printf("%d\t%d\n",a,i);
           if(flag==1){

                return NULL;
}
           V(mutex,1);
    }
        return NULL;
}
```

4、实验结果

编程完后调试无错误信息后开始测试，测试结果希望线程 1 可以用来计算 1 加到 100 的值，线程 2 可以打印中间的计算结果：

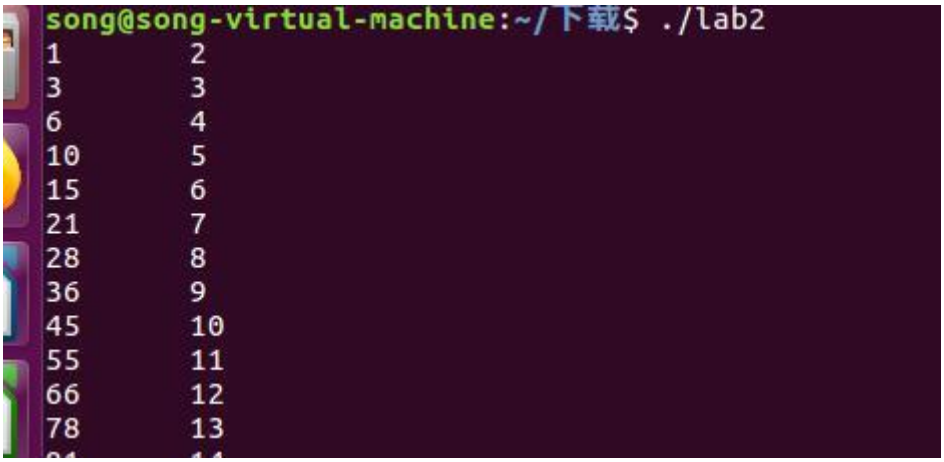

图 1-2.1 实验结果 1

图 2-2.1 实验结果 2

由结果可知，测试正确，每次加 1 最后在 i=101 时设置一个条件判断语句跳出进程，没执行一次线程 1 由于信号灯的控制，会阻止其继续执行。而执行线程 2. 线程二的执行过程同理。

三、实验心得

通过此次实验掌握了 Linux 下线程的概念，线程是一种特殊的进程，有时候也可以理解为进程的一部分。他们都是动态的，存在生命周期。此次实验之后更加清楚了程序和线程，进程间的区别。

通过这次实验了解了 Linux 线程同步与通信的主要机制，通过信号灯的互斥操作进行。实验过程中一开始创建了 1 个信号灯，因为觉得这样可以节约及资源，但是我忽略了一个很重要的问题，1 个信号只适用于线程是顺序执行的，而此次实验中线程会进行循环，所以一个信号灯会导致计算一次后直接打印很多次，然后等待线程 1 的 i 达到 101 时才会继续往下执行。经过改正后中古得出正确的结果。

这次试验也了解了通过信号灯操作实现线程间的同步与互斥。同步之处在于每次打印的结果都是上一次计算的结果。而互斥之处在于每次要往下计算时必须等待直到打印结束。也为实验 3 打下了基础。

实验三：共享内存与进程同步

一、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；

2、掌握 Linux 下进程同步与通信的主要机制。

二、实验内容

1、程序要求

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。

2、运行环境

软件配置（含操作系统版本）：Ubuntu 14.1.2 build-8497320

VMware® Workstation 14 Pro

硬件：内存 2GB

处理器 1

硬盘（SCSI） 20GB

网络适配器 NAT

任意监视器最大分辨率 2560*1600

3、源程序

```
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include<sys/stat.h>
#include <signal.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<fcntl.h>
#include <sys/wait.h>
int shmid=23333;
int Mykey=7777;
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;
}
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *arry;
};

int isempty;
int pid1,pid2;
int isfull;
int mutex;
int  isend;
char *s;
int main(int argc,char *argv[])
{
    int shm;
    //if(argc!=3){
       // printf("argc number fault!\n");
     //   exit(0);
   // }
    int readfile = open("read.txt",O_RDONLY,S_IRUSR|S_IWUSR);
    int writefile = open("write.txt",O_CREAT|O_RDWR,S_IRUSR|S_IWUSR);
    if(readfile==-1||writefile==-1){
        printf("file name error!\n");
        exit(0);
    }
    if((shm=shmget(shmid,10,IPC_CREAT|0666))==-1)
        printf("shm create error\n");
    s = (char *)shmat(shm,NULL,0);
    if((isempty=semget(Mykey,1,IPC_CREAT|0666))==-1)
        printf("semget error\n");
    if((isfull=semget(Mykey+1,1,IPC_CREAT|0666))==-1)
        printf("semget error\n");
    if((mutex=semget(Mykey+2,1,IPC_CREAT|0666))==-1)
        printf("semget error\n");
    if((isend=semget(Mykey+3,1,IPC_CREAT|0666))==-1)
        printf("semget error\n");
    union semun sem_union1;
```

```
sem_union1.val = 8;
if (semctl(isfull, 0, SETVAL, sem_union1) == -1){//initial the traffic1
    printf("seminit error\n");
}
sem_union1.val = 0;
if (semctl(isempty, 0, SETVAL, sem_union1) == -1){//initial the traffic2
    printf("seminit error\n");
}
sem_union1.val = 1;
if (semctl(mutex, 0, SETVAL, sem_union1) == -1){//initial the mutex
    printf("seminit error\n");
}
if (semctl(isend, 0, SETVAL, sem_union1) == -1){//initial the mutex
    printf("seminit error\n");
}
int i=0,j=0;
int flag=0;
if((pid1=fork())==0){
    char get;
    while(read(readfile,&get,sizeof(char))!=0){
        P(isfull,0);
        P(mutex,0);
        i=i%8;//取余
        s[i]=get;//读内容到缓存
        i++;//循环加 1
        V(mutex,0);
        V(isempty,0);
    }
    P(isend,0);
    i=i-1;
    i=i%8;
    s[8]=i;
    flag=1;
    s[9]=flag;
    V(isend,0);
    printf("read end!\n");
    exit(0);
}
else{
    if((pid2=fork())==0){
          while(1){
            P(isend,0);
            if(s[9]==1&&s[8]==j){
                V(isend,0);
```

```
                char g;
                g=s[j];
            //printf("%d\n",g);
                if(write(writefile,&g,1)==-1){
                    printf("%d\n",j);
                //printf("write error!\n");
                }//将当前缓存中的数据写入文件中
                printf("write end!\n");
                //break;
                exit(0);
            }
            V(isend,0);
            P(isempty,0);
            P(mutex,0);
            char g;
            j=j%8;//取余
            g=s[j];
            //printf("%d\n",g);
            if(write(writefile,&g,1)==-1){
                printf("%d\n",j);
                //printf("write error!\n");
            }//将当前缓存中的数据写入文件中
            //printf("%d\n",j);
            j++;//循环加 1
            V(mutex,0);
            V(isfull,0);
        }
        printf("write end!\n");
        exit(0);
    }
}
int p1state,p2state;
waitpid(pid1,&p1state,0);
waitpid(pid2,&p2state,0);
if (semctl(isempty, 0, IPC_RMID, sem_union1) == -1){
    printf("Failed to delete traffic1\n");
}

if (semctl(isfull, 0, IPC_RMID, sem_union1) == -1){
    printf("Failed to delete traffic2\n");
}
if (semctl(mutex, 0, IPC_RMID, sem_union1) == -1){
    printf("Failed to delete traffic2\n");
}
```

```
    shmdt((void *)s);
    shmctl(shm,IPC_RMID,0);
printf("suceed to delete mutex and shmid\n");
    printf("copy success!\n");
    close(writefile);
    close(readfile);
//   free(s);
    return 0;
}
```

4、实验结果

编写完成后，调试无误可以进行测试，测试结果希望实现一个复制，粘贴的功能，换种说法就是将文件 1 中的内容通过缓冲区写入文件 2 中。测试结果如下：

初始 write.txt 中内容为空，通过程序运行后可以显示 read.txt 中内容

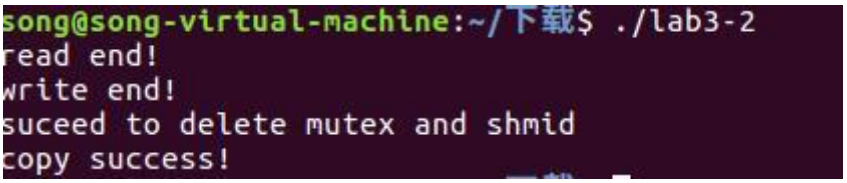

图 1-3.1 实验结果

此时对比 read.txt 和 write.txt：

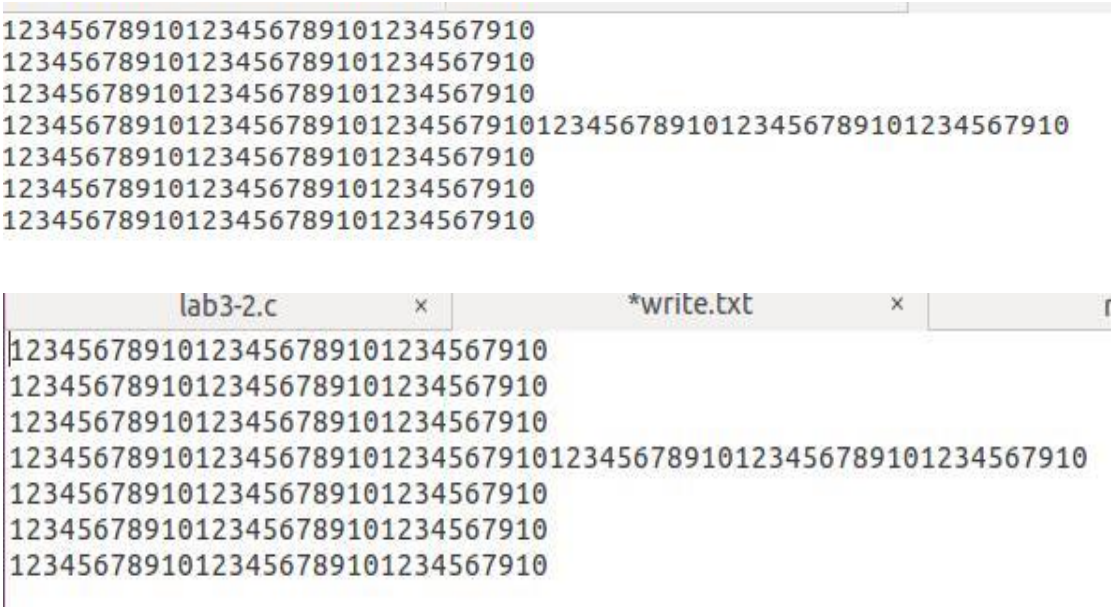

图 2-3.2 实验结果

可以看到两个文件内容相同，实验了内容的誊抄。结果正确。

三、实验心得

通过这次实验掌握了 Linux 下共享内存的概念与使用方法。共享内存，顾名思义就是通过一块小的存储区，在信号灯的控制下，可以每次将源文件内容放

到缓冲区中，再将缓冲区的内容写入到目标文件。在使用共享缓冲区时需要注意的是在完全读完文件内容和写完缓冲区时不要关闭输入输出流，其次是注意信号灯的控制，个人觉得应该需要三个信号灯，两个用于同步，一个用于互斥，互斥之处在于在每次读入内容时，为了防止从缓冲区写内容到目标文件，加一个互斥灯。同步之处在于设置缓冲区的大小和每次读入的大小，在缓冲区未满时由两个灯对缓冲区控制。

此次实验也掌握环形缓冲的结构与使用方法，创建缓冲区的时候需要注意的是赋予缓冲区权限，使得它可以用来作为共享内存，这一点和创建信号灯相似。环形缓冲区的循环使用之处在于在缓冲区的末尾设置一个标志位，每次到该标志位时就需停止，换到另外一个线程继续执行。

这次实验更加深入掌握 Linux 下进程同步与通信的主要机制。P 操作和 V 操作用来控制进程相当方便，可以有效的保证程序按照要求执行下去，但是需要注意的是一些变量有时候是需要设置为全局变量的。这次实验也为第四次实验打下了基础。

实验四：LINUX 文件目录

一、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。

二、实验内容

1、程序要求

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。

2、运行环境

软件配置（含操作系统版本）：Ubuntu 14.1.2 build-8497320

VMware® Workstation 14 Pro

硬件：内存 2GB

处理器 1

硬盘（SCSI）　　20GB

网络适配器　　NAT

任意监视器最大分辨率 2560*1600

3、源程序

```
#include<unistd.h>
#include<sys/stat.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<dirent.h>
#include <locale.h>
#include<time.h>
#include <pwd.h>
#include <grp.h>
#define N 256

void list_files(char *fullpath,char *path);
void print_status(char *path,int link_num,int size_num);
int comp(const void *a,const void *b);

int main(int argc,char* argv[])
{
    if(argc!=2){
        printf("enter two parameters\n");
        return 0;
    }
        char s[6];
        scanf("%s",s);
        while(strcmp(s,"ls-lr")!=0)
        {
            getchar();
            scanf("%s",s);
          }
    setlocale(LC_COLLATE, "");  //set sorting order
    char *path;
    path=(char*)malloc(sizeof(argv[1])+1);
    int link=0;
    int size=0;
    strcpy(path,argv[1]);
//  path[strlen(argv[1])+1]='\0';

    struct stat statbuf;
    if(stat(path,&statbuf)==-1)
    {
```

```
        printf("error fliename\n");
        return 0;
    }
    if(S_ISDIR(statbuf.st_mode)) //判断是否为目录
        list_files(path,path);
    else{
       for (; statbuf.st_nlink; statbuf.st_nlink /= 10)
           link++;
       for (; statbuf.st_size; statbuf.st_size /= 10)
           size++;
       print_status(path, link, size);
    }
    return 0;
}
void list_files(char *fullpath,char*path)
{
    struct dirent *entry = NULL;
    struct stat s_buf;
    DIR *dir=opendir(path);
    if(dir==NULL)
    {
        printf("fail to open path\n");
        exit(0);
    }
    chdir(path);
    if(strcmp(path,fullpath)!=0)
        putchar('\n');
    printf("%s:\n",fullpath);
    int entry_count=0;
    while((entry=readdir(dir))!=NULL)
    {
        if(strncmp(entry->d_name,".",1)==0||strncmp(entry->d_name,"..",2)==0)
continue ; //比较是否子串 是跳过
        entry_count++;
    }

    int pos=0;
    rewinddir(dir);
    struct dirent *entry_bufs = (struct dirent*)malloc(sizeof(struct dirent) *
entry_count); //存放文件信息
    while((entry=readdir(dir))!=NULL)
    {
        if(strncmp(entry->d_name,".",1)==0||strncmp(entry->d_name,"..",2)==0)
continue;
```

```
        memcpy(entry_bufs+pos,entry,sizeof(struct dirent));
        pos++;
      }
      qsort(entry_bufs,entry_count,sizeof(struct dirent),comp);

      unsigned long total = 0;
      int link_max = 0, size_max = 0;
      int link_num = 0, size_num = 0;
      for(int i=0;i<entry_count;i++)
      {
        entry=entry_bufs+i;
        stat(entry->d_name,&s_buf); //保存文件信息到 s_buf
        if(link_max<s_buf.st_nlink)
          link_max=s_buf.st_nlink;
        if(size_max<s_buf.st_size)
          size_max=s_buf.st_size;
        total+=s_buf.st_blocks;
       }
       for (; link_max; link_max /= 10)
          link_num++;
       for (; size_max; size_max /= 10)
          size_num++;

     printf("total %lu documents\n",total*512/1024);
     for(int i=0;i<entry_count;i++)
     {
         entry=entry_bufs+i;
         print_status(entry->d_name, link_num, size_num); //print 文件信息
     }
     //递归
     for(int i=0;i<entry_count;i++)
     {
         entry=entry_bufs+i;
         char    *new_path    =    (char*)malloc(sizeof(*fullpath)    +    2    +
sizeof(entry->d_name)); //下一个目录
         strcpy(new_path,fullpath);
         strcat(new_path,"/");
         strcat(new_path,entry->d_name);
         if(entry->d_type==4)  //字符设备 DT_CHR
            list_files(new_path,entry->d_name); //递归下一个文件
      }
    chdir("..");
    closedir(dir);
}
```

```
int comp(const void *a,const void *b)
{
        struct dirent *dir_a=(struct dirent*)a;
        struct dirent *dir_b=(struct dirent*)b;
        return strcoll(dir_a->d_name,dir_b->d_name);// 排序中文文件
}

void print_status(char *path,int link_num,int size_num)
{
    struct stat s_buf;
    char *time;
    struct passwd *pwd_buf;
    struct group *group_buf;
    char modstr[11];
    stat(path,&s_buf);
    strcpy(modstr,"----------");
      if (S_ISDIR(s_buf.st_mode))
      modstr[0] = 'd';
    if (S_ISCHR(s_buf.st_mode))
        modstr[0] = 'c';
    if (S_ISBLK(s_buf.st_mode))
        modstr[0] = 'b';
  if (S_ISFIFO(s_buf.st_mode))
    modstr[0] = 'p';
  /* User mod */
    if (s_buf.st_mode & S_IRUSR)
        modstr[1] = 'r';
    if (s_buf.st_mode & S_IWUSR)
        modstr[2] = 'w';
    if (s_buf.st_mode & S_IXUSR)
        modstr[3] = 'x';
  /* Group mod */
    if (s_buf.st_mode & S_IRGRP)
        modstr[4] = 'r';
    if (s_buf.st_mode & S_IWGRP)
        modstr[5] = 'w';
    if (s_buf.st_mode & S_IXGRP)
        modstr[6] = 'x';
 /* Other mod */
    if (s_buf.st_mode & S_IROTH)
        modstr[7] = 'r';
    if (s_buf.st_mode & S_IWOTH)
        modstr[8] = 'w';
```

```
    if (s_buf.st_mode & S_IXOTH)
        modstr[9] = 'x';
    pwd_buf = getpwuid(s_buf.st_uid);
    group_buf = getgrgid(s_buf.st_gid);
    time=ctime(&s_buf.st_mtime);  //文件最后访问时间
    printf("%s %*d %s %s %*ld %.12s %-s\n", \
      modstr, link_num, (int)s_buf.st_nlink, pwd_buf->pw_name,
      group_buf->gr_name, size_num, (long)s_buf.st_size, time + 4, path);
}
```

4、实验结果

编写完程序调试之后可以直接进行测试，测试结果希望实现 1s -1r 功能，即能够展示文件目录信息：

首先运行程序后输入 1s-1r：

```
song@song-virtual-machine:~/下载$ ./lab4-1 /home/song
1s-1r
/home/song:
total 36 documents
drwxrwxr-x 3 song song 4096 Dec 27 10:26 demo
drwxr-xr-x 2 song song 4096 Jun  5 09:01 本地的
```

图 4-1.1 测试结果

可以看到目录/home/song 下总计 36 个文件，以下是对这些文件信息的是输出，包括权限信息，子目录个数，文件所有者名，文件大小，最后修改日期，和文件 /目录名：

```
/home/song/demo/U201614749:
total 132 documents
-rw-rw-r-- 1 song song 88992 Jun 18 11:30 bisassemble.txt
-rwxrwxr-x 1 song song 23108 Jun  5 09:09 bomb
-rwxrwxr-x 1 song song  4069 Jun  5 09:09 bomb.c
-rw-rw-r-- 1 song song    10 Jun  5 09:09 ID
drwxrwxr-x 2 song song  4096 Jun 19 20:26 lab3
-rw-rw-r-- 1 song song    70 Jun  5 09:09 README
-rw-rw-r-- 1 song song    75 Jun 18 16:24 无标题文档
```

图 4-1.2 测试结果

结果的对齐可以通过%*d 类似这样的内容跳过某些输出实现，最后运行结果正确，符合预期。

三、实验心得

通过本次实验了解 Linux 文件系统与目录操作，对于文件操作的几个函数有

了更深的理解，stat 函数作为作为获取文件信息的函数，其中的结构体 struct stat 里面的信息相当重要，是完成这次实验的核心所在。弄清楚文件权限信息的表示方法和所有者 ID 表示方法后实验就会比较简单了。

通过这次实验也了解 Linux 文件系统目录结构，这次试验相比前三次实验难度有所增加，需要课前准备够的知识否则这次实验会很艰难。通过过这次实验了解到了 linux 下文件的目录结构，可以藏书出信息看到文档中的目录级别和每级目录下韩寒有的目录级别。

这次实验之后掌握了文件和目录的程序设计方法，但是总体说来这次实验的收获并不比前三次实验多，首先文件目录这几个函数感觉也就在实验时用下，最后也会完全忘记，也许老师可以稍微修改一下实验 4 的内容，改为 linux 下进程和线程更进一步的探讨。继续增加对于前三次实验的理解和对线程进程的理解。

最后也希望操作系统这门课程越来越好。