实验 12　Linux 文件管理—文件的创建和读写

1. 实验目的

(1) 了解 Linux 文件系统中关于文件 inode 结点和文件描述符的概念。

(2) 掌握通过 read、write 系统调用以及文件描述符对文件进行读写操作方法。

2. 实验相关知识

(1) Linux 文件系统

Linux 文件系统由引导块、超级块、索引结点表、目录和文件的数据块四部分组成。其中引导块存储系统启动时所需的信息；超级块存储文件系统的信息，包括状态、类型、大小、区块数和索引结点数等，操作系统启动后，内核把其中内容复制到内存中，并周期性的利用内存里的最新内容去更新硬盘上的超级块中的内容。每一个存放在磁盘上的文件由索引结点和数据块两部分构成，数据块是实际存放文件数据的磁盘块。

(2) 索引结点 inode

一个结构，包含文件的长度、创建及修改时间、权限、所属关系、在磁盘中的物理块号等文件属性信息。其中文件在磁盘块区的物理地址是个非常重要的信息，操作系统就是依靠它在硬盘上定位相关的文件，并读取文件。

索引结点表：所有索引结点的一个线性表，并统一进行编号。0、1 号一般不用，根目录“/”对应 2 号索引结点；每个索引结点存储一个文件(除文件名)的属性信息。一个文件系统维护了一个索引结点数组，每个文件或目录都与索引结点数组中的唯一一个元素对应。索引结点号表示结点在数组中的下标。

磁盘上的文件分为目录文件和普通文件，目录文件的数据块存储该目录下文件的文件名与其 inode 结点的对应关系。

通过系统命令 ls 显示文件或者目录信息或使用系统调用 stat、fstat、lstat 的时候，就需要用到索引结点中的信息。ls 命令为了确定一个文件的 inode 结点需要查找它所在的目录，根据目录文件中该文件对应的 inode 结点号，读取它的 inode 结点信息，得到文件的属性。

(3) 系统打开文件链表

Linux 虚拟文件系统中有一个重要的数据结构，即系统打开文件链表。它是一个双向链表，系统全局变量 first_file 指向其表头，Linux 系统内核利用它对所有进程打开的文件集中管理。链表中每个结点是一个 file 结构，存放一个已打开文件的管理控制信息，进程访问文件之前必须用系统调用 open( )打开文件，这时系统建立一个 file 结构体，把它加入到系统打开文件链表中，并把这个 file 结构体的首地址写入进程文件描述符表 fd 第一个空闲元素中，用户在程序中使用文件描述符（又称文件标识号）访问文件。

当进程通过 fork( )创建一个子进程， 子进程共享父进程的打开文件表。父子进程两者的 fd 下标相同的两个元素指向同一个 file 结构。一个文件可以被某个进程多次打开，每次都分配一个 file，并占用该 fd 数组的一项，其下标就是文件标识号。 它们的 file 结构中的 f_inode 都指向同一个 inode。

(4) 进程的文件管理

进程打开的所有文件，由进程的两个私有结构管理。fs_struct：记录着文件系统根目录和当前目录；files_struct：包含着进程的打开文件表。

```
struct fs_struct {
    int count;       /共享此结构的计数值
    unsigned short umask;       //文件掩码
```

```
    struct inode    * root, * pwd;       //根目录和当前目录 inode 指针
    };
```

root：系统的根目录 inode， 在按照绝对路径访问文件时就从这个指针开始。pwd：指向当前目录 inode，相对路径从这个指针开始。在打开文件时，fs_struct 用于查找文件的 inode 结点的信息，以便向新建立的 file 结构中写入信息。

```
#define NR_OPEN 256
struct files_struct {
    int count;        //共享该结构的计数值
    fd_set     close_on_exec;
    fd_set     open_fds;
    struct file *    fd[NR_OPEN];
    };
```

fd 称为文件描述符表，每个元素是一个指向 file 结构体的指针，其下标就是文件标识号。当一个进程打开一个文件时，内核在该用户的文件描述符表和系统的文件表中各登记一个表项。其中文件描述符表中的表项含有指向文件表表项的指针，文件表表项中又含有指向文件 inode 索引结点的指针。

特殊文件描述符：进程开始运行时自动打开三个文件（文件描述符表前三项）。0：STDIN_FILENO，对应标准输入；1: STDOUT_FILENO，对应标准输出；2: STDERR_FILENO，对应标准错误输出。

3. 相关系统调用

(1) int open(const char *pathname, int flags);

int open(const char *pathname, int flags, mode_t mode) ;

功能：打开文件。

返回值：成功：一个文件描述符，以后对该文件的所有操作都通过这个文件描述符来实现；

失败：-1。

参数：pathname：要打开的文件名（包含路径名称，缺省认为在当前路径下）

flags：打开文件方式，可以取下面的一个值或者是几个值的组合：

标志	含义
O_RDONLY	以只读的方式打开文件
O_WRONLY	以只写的方式打开文件
O_RDWR	以读写的方式打开文件
O_APPEND	以追加的方式打开文件
O_CREAT	创建一个文件
O_EXEC	如果使用了 O_CREAT 而且文件已经存在，就会发生错误
O_NOBLOCK	以非阻塞的方式打开一个文件
O_TRUNC	如果文件已经存在，则删除文件的内容

O_RDONLY、O_WRONLY、O_RDWR 三个标志只能使用任意的一个。

以 O_CREAT 为标志的 open 实际上实现了文件创建的功能。下面的函数等同 creat()函数：

int open(pathname, O_CREAT | O_WRONLY | O_TRUNC, mode);

mode：文件的存取权限，可以是以下情况的组合：

标志	含义
S_IRUSR	用户可以读
S_IWUSR	用户可以写
S_IXUSR	用户可以执行

S_IRWXU	用户可以读、写、执行
S_IRGRP	组可以读
S_IWGRP	组可以写
S_IXGRP	组可以执行
S_IRWXG	组可以读写执行
S_IROTH	其他人可以读
S_IWOTH	其他人可以写
S_IXOTH	其他人可以执行
S_IRWXO	其他人可以读、写、执行
S_ISUID	设置用户执行 ID
S_ISGID	设置组的执行 ID

除了可以通过上述宏进行“或”逻辑产生标志以外，也可以自己用数字来表示。Linux 总共用 5 位数字来表示文件的各种权限：第一位表示设置用户 ID；第二位表示设置组 ID；第三位表示用户自己的权限位；第四位表示组的权限；最后一位表示其他人的权限。每个数字可以取 1（执行权限）、2（写权限）、4（读权限）、0（无）或者是这些值的和。例如，要创建一个用户可读、可写、可执行，但是组没有权限，其他人可以读、可以执行的文件，并设置用户 ID 位。那么，应该使用的模式是 1（设置用户 ID）、0（不设置组 ID）、7（1+2+4，读、写、执行）、0（没有权限）、5（1+4，读、执行）即 10705。

open(“test”, O_CREAT, 10705);　等价于：

open(“test”, O_CREAT, S_IRWXU | S_IROTH | S_IXOTH | S_ISUID) ;

(2) int close(int fd) ;

功能：关闭文件。关闭文件时，内核对文件在系统打开文件表中的引用计数减 1，如果减为 0，则释放该文件描述项，使其为空闲可用项；对文件 inode 索引结点中的引用计数减 1；释放该文件的文件描述符。当一个进程终止时，内核会自动检查并回收该进程所有的文件描述符，用户不必显式地调用 close。

返回值：成功：0；

失败：-1。

参数：fd：要关闭的文件的文件描述符。

(3) int read(int fd, const void *buf, size_t length);

功能：从文件描述符 fd 所指定的文件中读取 length 个字节到 buf 所指向的缓冲区中。

返回值：实际读入的字节数。

参数：fd：文件描述符。

buf：指向缓冲区的指针。

length：缓冲区的大小（以字节为单位）。

(4) int write(int fd, const void *buf, size_t length) ;

功能：把 length 个字节从 buf 指向的缓冲区中写到文件描述符 fd 所指向的文件中。

返回值：实际写入的字节数。

参数：fd：文件描述符。

buf：指向缓冲区的指针。

length：缓冲区的大小（以字节为单位）。

(5) int dup(int oldfd);

int dup2(int oldfd, int newfd);

功能：复制一个现存的文件描述符，使两个文件描述符指向同一个 file 结构体。经常用来重定向进程的 stdin、stdout 和 stderr。

dup: 复制已打开的文件描述符。

dup2：按指定条件复制文件描述符。

返回值：成功：新分配或指定的文件描述符；

出错：-1。

参数：oldfd：现存的文件文件描述符。

newfd：指定新描述符的数值。

dup2 可以用 newfd 参数指定新描述符的数值。如果 newfd 当前已经打开，则先将其关闭再做 dup2 操作，如果 oldfd 等于 newfd，则 dup2 直接返回 newfd 而不用先关闭 newfd 再复制。

newfd=dup(oldfd);　　// newfd 是系统分配的、未使用的最小描述符

dup2(oldfd，newfd);　　//newfd 自己指定结果都是让 newfd 指向 oldfd。

分析下面一段代码：

```
int oldfd;
oldfd = open(“app_log”, (O_RDWR | O_CREATE), 0644 );
dup2( oldfd, 1 );
close( oldfd );
```

打开了一个新文件，称为“app_log”，并得到一个文件描述符 oldfd。调用 dup2 函数，参数为 oldfd 和 1，oldfd 替换由 1 代表的文件描述符（即 stdout，因为标准输出文件的 id 为 1）。任何写到 stdout 的东西，现在都写入名为“app_log”的文件中。

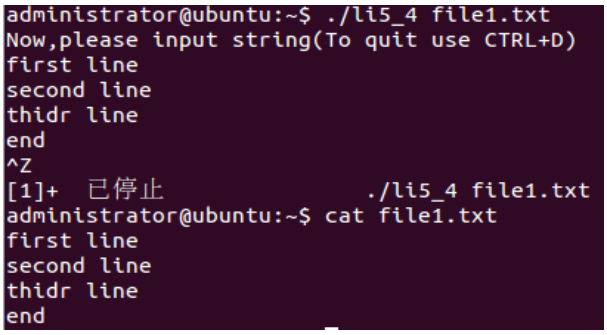
4. 实验程序

【程序 5_4】从键盘输入串，写到文件中。

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#define BUFFER_SIZE 1024
int main(int argc,char **argv)
{
    int fd; char buffer[BUFFER_SIZE];
    if(argc!=2)
    {
        fprintf(stderr,"Usage:%s outfilename\n\a",argv[0]);
        exit(1);
    }
    if((fd=open(argv[1],O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR))==-1)
    {
        fprintf(stderr,"Open %s Error:%s\n\a",argv[1],strerror(errno));
        exit(1);
    }
```

```
    printf("Now,please input string");
    printf("(To quit use CTRL+D)\n");
    while(1)
    {
        fgets(buffer,BUFFER_SIZE,stdin);
        if(feof(stdin))
            break;
        write(fd,buffer,strlen(buffer));
    }
    close(fd);
    exit(0);
}
```

【程序运行结果截图】

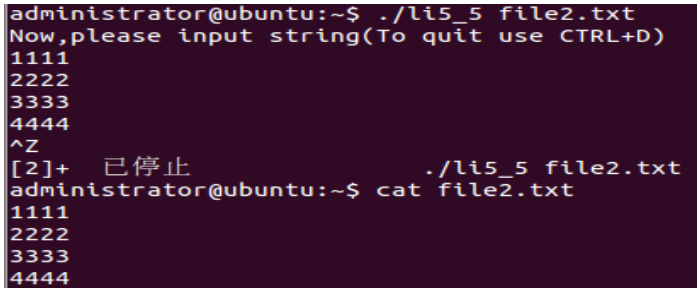


【程序 5_5】使用 dup2 将标准输出重定向。从键盘输入串，写到文件中。

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#define BUFFER_SIZE 1024
int main(int argc,char **argv)
{
    int fd;
    char buffer[BUFFER_SIZE];
    if(argc!=2)
    {
        fprintf(stderr,"Usage:%s outfilename\n\a",argv[0]);
        exit(1);
    }
    if((fd=open(argv[1],O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR))==-1)
    {
        fprintf(stderr,"Open %s Error:%s\n\a",argv[1],strerror(errno));
```

```
        exit(1);
    }
    if(dup2(fd,STDOUT_FILENO)==-1)
    {
        fprintf(stderr,"Redirect Standard Out Error:%s\n\a",strerror(errno));
        exit(1);
    }
    fprintf(stderr,"Now,please input string");
    fprintf(stderr,"(To quit use CTRL+D)\n");
    while(1)
    {
        fgets(buffer,BUFFER_SIZE,stdin);
        if(feof(stdin))
            break;
        write(STDOUT_FILENO,buffer,strlen(buffer));        //将 buffer 中串向 fd 文件中写
    }
    close(fd);
    exit(0);
}
```

【程序运行结果截图】

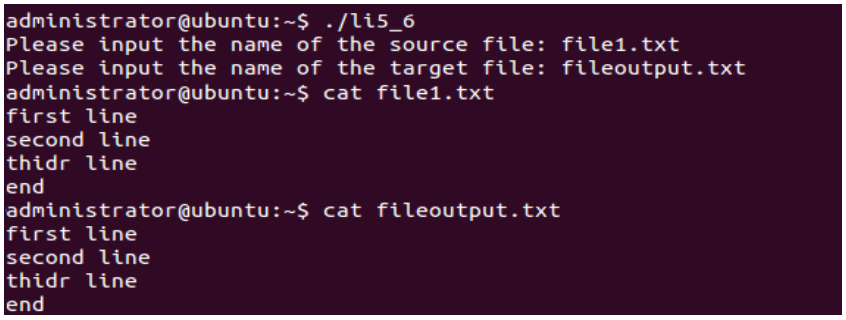


【程序 5_6】简单的文件拷贝。源文件是只有一个字符串构成的文本文件。

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#define Length 1024
int main( )
{
    int fdw, fdr, len;    char str[Length];    char sourcename[15],targetname[20];
    printf("Please input the name of the source file: ");
    gets(sourcename);
    printf("Please input the name of the target file: ");
    gets(targetname);
    fdr=open(sourcename,O_RDONLY);
    if (fdr)    len=read(fdr,str,Length);
    else
    {
        printf("read file error");
```

```
        exit(0);
    }
    fdw=open(targetname,O_CREAT|O_RDWR);
    write(fdw,str,len);
    close(fdr);
    close(fdw);
}
```

【程序运行结果截图】



实验 13　Linux 文件管理—文件的定位

1. 实验目的

掌握通过 lseek 系统调用移动文件读写指针的方法，从而实现对文件的随机读写。

2. 相关系统调用

int lseek(int fd, offset_t offset, int whence);

功能：将文件读写指针相对 whence 移动 offset 个字节。

返回值：文件指针相对于文件头的位置。

参数：fd：文件描述符；

offset：移动的字节数，可取负值；

whence：SEEK_SET：相对文件开头

SEEK_CUR：相对文件读写指针的当前位置

SEEK_END：相对文件末尾

如 lseek(fd, -5, SEEK_CUR);　　//将文件指针相对当前位置向前移动 5 个字节

如 lseek(fd, 0, SEEK_END);　　//返回值是文件的长度

3. 实验程序

【程序 5_7】文件的定位操作，并用 od 命令查看文件的实际内容。

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#define NEWFILE (O_WRONLY|O_CREAT|O_TRUNC)
int main(void)
{
    char buf1[]={"abcdefghijk"};
    char buf2[]={"1234567890"};
    int fd;
    int length;
    if((fd=open("test", NEWFILE,0600))==-1)
    {
        printf("ERROR, open write file error:%s\n", strerror(errno));
        exit(255);
    }
    length=strlen(buf1);
    if(write(fd,buf1,length)!=length)
    {
        printf("ERROR, write file failed: %s\n", strerror(errno));
        exit(255);
    }
    if(lseek(fd,80,SEEK_SET)==-1)        //将读写指针移动到相对文件头 80 个字符的位置
    {
        printf("ERROR, lseek failed: %s\n", strerror(errno));
        exit(255);
    }
    length=strlen(buf2);
    if(write(fd,buf2,length)!=length)
    {
        printf("ERROR, write file failed: %s\n", strerror(errno));
        exit(255);
    }
    close(fd);
    return 0;
}
```

4. 【程序运行结果截图及运行说明】

程序运行后，如果直接用文本编辑器打开此文件，则看不到文件读写指针移动的效果。这时需要使用系统命令 od。它用来查看特殊格式的文件内容，对于访问或可视地检查文件中不能直接显示在终端上的字符很有用。

命令格式：od ［选项］ 文件

功能：读取指定文件的内容，并将其内容以八进制码呈现出来。

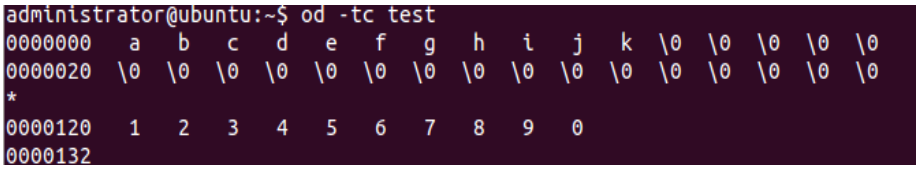
参数：-A 指定地址基数。

d：十进制； o：八进制(系统默认值)； x: 十六进制

- t：指定数据的显示格式

c：ASCII 字符或反斜杠序列； d：有符号十进制数； f：浮点数

o：八进制(系统默认值为 02)； u：无符号十进制数； x：十六进制数



$od –tc test

按地址基数为 8 进制，以 ASCII 字符或反斜杠序列显示文件 test 内容。

在文件 test 中，八进制 0000120 位置存放的是字符'1'，它的位置距离文件头的偏移量是 $(120)_8$，转换成十进制是 80。这是因为在写串"1234567890"之前，用 lseek(fd,80,SEEK_SET) 将文件读写指针的位置移动到相对文件头 80 个字符的位置。

实验 14　Linux 文件管理—文件状态信息的获取

1. 实验目的

(1) 掌握 Linux 文件系统中关于文件 inode 结点和文件描述符的相关知识。

(2) 掌握通过 stat 系统调用从程序中访问和显示文件元数据的方法。

2. 相关系统调用

(1) int fstat(int filedes,struct stat *buf);

(2) int stat(const char *path,struct stat *buf);

(3) int lstat(const char *path,struct stat *buf);

功能：通过路径或文件描述符得到文件信息，这些信息被写到结构 struct stat 的缓冲区中。

返回值：成功：0；

　　出错：-1。

参数：path：文件路径；

　　buf：返回的文件信息（struct stat）。

```
struct stat{
    dev_t       st_dev;     //文件所在的设备 ID
      ino_t       st_ino;       //i 结点号
    mode_t      st_mode;    //文件对应的模式:文件、目录
    nlink_t     st_nlink;   //硬链接个数
      uid_t       st_uid;     //uid
    gid_t       st_gid;     //gid
      dev_t       st_rdev;    //设备文件 ID（如果是特殊文件）
      off_t       st_size;    //普通文件，对应的文件字节数
    time_t      st_atime;   //文件最后被访问的时间
    time_t      st_mtime;    //文件内容最后被修改的时间
    time_t      st_ctime;   //文件状态改变时间
      blksize_t   st_blksize;   //块大小（字节）
    blkcnt_t    st_blocks;    //文件块个数
    };
```

其中能直接显示的信息有：i 结点号、链接数、所有者 id、所属组 id、文件大小、文件系统磁盘块大小、分配的磁盘块个数等。经过转换才能看到的信息有：文件类型、权限、时间等。

3. 实验程序

【程序 5_8】从 struct stat 中获取文件的 time of last access 信息，并用 char *ctime（time_t* t）转换成可读形式。

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
int main(int args,char *argv[])
{
    struct stat     statbuf;
    time_t   t;
    if(args < 2)
    {
        printf("please input a file\n");
        exit(1);
    }
    if( stat(argv[1],&statbuf) == 0)
    {
        t = statbuf.st_atime;
        printf("%s\n",ctime(&t));
    }
    else
    {
        perror(argv[1]);
        exit(1);
    }
    return 0;
}
```

4. 【程序运行结果截图】

```
administrator@ubuntu:~$ ./li5_8 file1.txt
Thu Feb 16 19:51:23 2012

administrator@ubuntu:~$ ./li5_8 li5_8.c
Thu Feb 16 19:24:06 2012
```

5. 实验思考与扩充

(1) 程序说明：perror ( )：将上一个函数发生错误的原因输出到 stderr 。实参所指的字符串会先打印出，后面再加上错误原因字符串。系统依照全局变量 errno 的值来决定要输出的字符串。

(2) 扩充程序，除文件最后被访问的时间外，读取并显示文件的其他属性信息。

实验 15　Linux 文件管理—文件目录操作

1. 实验目的

(1) 掌握通过 stat 系统调用从程序中访问和显示文件元数据的方法。

(2) 掌握通过 readdir 系统调用读取目录文件信息的方法。

2. 相关系统调用

(1) DIR *opendir(const char *path);

功能：打开目录。

返回值：成功：DIR 指针；

出错：NULL。

参数：path：目录名。

(2) int closedir(DIR *dirp);

功能：关闭目录。

返回值：成功： 0；

出错：-1。

参数： dirp：opendir 返回的 DIR 指针。

(3) struct dirent *readdir(DIR *dir);

功能：读取目录文件内容。

返回值：成功：dirent 指针，即参数 dir 目录下的目录进入点；出错或读取到目录文件尾：NULL。

参数：dirp：opendir 返回的 DIR 指针。

结构 dirent 定义如下：

```
struct dirent {
    long d_ino;       //文件对应  inode number
    off_t d_off;      //目录项在目录文件中偏移量
    unsigned short d_reclen;      //文件名长度 len of d_name
    char d_name [NAME_MAX+1];     //文件名 file name
    }
```

3. 实验程序

【程序 5_9】程序有一个参数。如果参数是一个文件名， 输出这个文件的大小和最后修改的时间，如果是一个目录，输出此目录下所有文件的大小和修改时间。

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
static int get_file_size_time(const char *filename)
{
    struct stat statbuf;
    if(stat(filename,&statbuf)==-1)      //取 filename 的状态
```

```
    {
        printf("Get stat on %s Error:%s\n",filename,strerror(errno));
        return(-1);
    }
    if(S_ISDIR(statbuf.st_mode))
        return(1);      //判断是否是目录文件
    if(S_ISREG(statbuf.st_mode))      //判断是否是普通文件
         printf("%ssize:%ldbytes\tmodifiedat%s",filename,statbuf.st_size,ctime(&statbuf.st_mti
     me));       //输出文件的大小和最后修改时间
    return(0);
}
int main(int argc,char **argv)
{
    DIR *dirp;
    struct dirent *direntp;
    int stats;
    char path[1024];
    if(argc!=2)      //判断输入是否为两个参数
    {
        printf("Usage:%s filename\n\a",argv[0]);
        exit(1);
    }
    if(((stats=get_file_size_time(argv[1]))==0)||(stats==-1))
        exit(1);       //若 argv[1]为文件（输出文件大小和最后修改时间）或者 stats 错误，
都退出
    if((dirp=opendir(argv[1]))==NULL)      //打开目录，将打开的目录信息放至 dirp 中，若
为空，则打开失败
    {
        printf("Open Directory %s Error:%s\n",argv[1],strerror(errno));
        exit(1);
    }
    while((direntp = readdir(dirp)) != NULL)
    {
        memset(path, 0, 1024);
        strcpy(path, argv[1]);
        strcat(path, "/");
        strcat(path, direntp -> d_name);
        if(get_file_size_time(path)==-1)      //读取 dirp 目录下文件，直到出错或结束，退出
            break;
    }
    closedir(dirp);
    exit(1);
}
```

4.【程序运行结果截图】

```
administrator@ubuntu:~$ pwd
/home/administrator
administrator@ubuntu:~$ ls -l
总用量 176
drwxrwxr-x 2 administrator administrator 4096 2012-02-16 18:54 3_process
drwxrwxr-x 2 administrator administrator 4096 2012-02-24 14:30 5_文件管理
drwxrwxr-x 3 administrator administrator 4096 2012-02-23 19:26 d1
-rw-r--r-- 1 administrator administrator  344 2012-02-16 19:20 li5_8.c~
-rwxrwxr-x 1 administrator administrator 7634 2012-02-24 14:35 li5_9
-rw-r--r-- 1 administrator administrator 1375 2012-02-24 14:22 li5_9.c
-rw-r--r-- 1 administrator administrator  492 2012-02-16 07:53 malloc.c~
drwxrwxr-x 2 administrator administrator 4096 2011-11-02 11:28 Ubuntu One
-rw-r--r-- 1 administrator administrator   83 2011-11-03 19:08 w.c~
drwxr-xr-x 2 administrator administrator 4096 2011-10-23 12:36 公共
drwxr-xr-x 2 administrator administrator 4096 2011-10-23 12:36 模板
drwxr-xr-x 2 administrator administrator 4096 2011-10-23 12:36 视频
drwxr-xr-x 2 administrator administrator 4096 2012-02-24 14:55 图片
-rw-r--r-- 1 administrator administrator 1029 2012-02-16 16:41 li3_16.c~
-rwxrwxr-x 1 administrator administrator 7490 2012-02-16 15:51 li3_17
```

```
administrator@ubuntu:~/d1$ pwd
/home/administrator/d1
administrator@ubuntu:~/d1$ ls -l
总用量 16
drwxrwxr-x 2 administrator administrator 4096 2012-02-23 19:22 d2
-rw-r--r-- 1 administrator administrator 1364 2012-02-16 15:21 li3_13.c
-rwxrwxr-x 1 administrator administrator 7490 2012-02-16 15:51 li3_17
-rw-rw-r-- 1 administrator administrator    0 2012-02-23 19:21 wenben.txt~
administrator@ubuntu:~/d1$ cd ..
administrator@ubuntu:~$ pwd
/home/administrator
administrator@ubuntu:~$ ./li5_9 d1
d1/li3_13.c size:1364 bytes     modified at Thu Feb 16 15:21:29 2012
d1/wenben.txt~ size:0 bytes     modified at Thu Feb 23 19:21:03 2012
d1/li3_17 size:7490 bytes       modified at Thu Feb 16 15:51:49 2012
```

```
administrator@ubuntu:~$ pwd
/home/administrator
administrator@ubuntu:~$ ./li5_9 li5_9.c
li5_9.c size:1375 bytes modified at Fri Feb 24 14:22:28 2012
administrator@ubuntu:~$ ./li5_9 li5_9
li5_9 size:7634 bytes   modified at Fri Feb 24 14:35:04 2012
```

5. 实验思考与扩充

(1) 在程序中增加代码，完成下面功能：如果目录下有“source.txt”文件，就将其内容复制到名为”target.txt”的文件中。（参照【程序 5_6】）。在自己主目录“/home/xxxxxx”下建立一个名为“source.txt”的文本文件，由一个字符串构成。以“/home/xxxxxx”为参数运行程序，然后用 cat 命令将”target.txt”内容打印出来。以验证是否正确完成拷贝。

(2) 还可对程序功能进行进一步扩充，显示文件的其他属性信息，如 i 结点号、文件的物理块数等。