

# 《计算机操作系统》实验报告

---

## 实验题目：Linux 文件实验

姓名：严昕宇      学号：20121802      实验日期：2022.12.30

---

### 实验环境：

实验设备：Lenovo Thinkbook16+ 2022

操作系统：Ubuntu 22.04.1 LTS 64 位

### 实验目的：

1. 掌握操作系统中文件分类的概念
2. 了解 Linux 文件系统管理文件的基本方式和特点
3. 学会使用 Linux 文件系统的命令界面和程序界面的基本要领

### 实验准备：

1. 复习操作系统中有关文件系统的知识，熟悉文件的类型、i 节点、文件属性、文件系统操作等概念。
2. 熟悉《实验指导》第五部分“文件系统的系统调用”。了解 Linux 文件系统的特点、分类。
3. 阅读例程中给出的相应的程序段。

### 实验方法：

运行命令界面的各命令并观察结果。

用 vi 编写 c 程序(假定程序文件名为 prog1.c)

编译程序

\$ gcc -o prog1.o prog1.c 或 \$ cc -o prog1.o prog1.c

运行

\$/prog1.o

观察运行结果并讨论。

### 实验内容：

1. 用 shell 命令查看 Linux 文件类型
2. 用 shell 命令了解 Linux 文件系统的目录结构
3. 用命令分别建立硬链接文件和符号链接文件。通过 ls -il 命令所示的 inode、链接
4. 复习 Unix 或 Linux 文件目录信息 i 节点的概念。编程察看指定文件的 inode 信息
5. 修改父进程创建子进程的程序，用显示程序段、数据段地址的方法，说明子进程继承父进程的所有资源。再用父进程创建子进程，子进程调用其它程序的方法进一步证明子进程执行其它程序时，程序段发生的变化。
6. 编写一个涉及流文件的程序

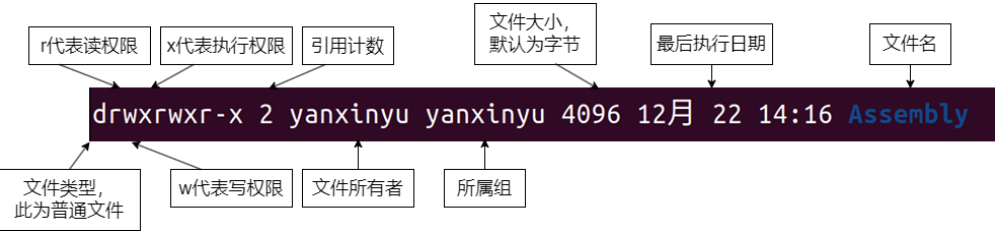
# 操作过程 1:

【操作要求 1】用 shell 命令查看 Linux 文件类型

# 结果 1:

```
yanxinyu@ThinkBook16-2022:~/Code$ ls -l
总用量 12
drwxrwxr-x 2 yanxinyu yanxinyu 4096 12月 22 14:16 Assembly
-rw-rw-r-- 1 yanxinyu yanxinyu 158 12月 22 14:00 Philo.cpp.gz
-rw-rw-r-- 1 yanxinyu yanxinyu 1824 12月 22 14:00 ProcessScheduler.cpp.gz
```

其中第一个字母包含了该文件的文件类型.具体含义如下:



思考: Linux 文件类型有哪些? 用什么符号表示。

答: Linux 中文件类型有正规文件(包括纯文件文件(ASCII)、二进制文件(binary)、数据格式文件(data))、目录文件、设备文件(块设备, 字符设备)、链接文件(硬链接, 软链接)、有名管道文件和套接字文件。

文件属性	文件类型
-	正规文件, File
d	目录文件
b	Block Device, 即块设备文件; 如硬盘支持以 Block 为单位进行随机访问
c	Character Device, 即字符设备文件; 如键盘支持以 Character 为单位进行线性访问
l	Symbolic Link, 即符号链接文件, 又称软链接文件
p	Pipe, 即命名管道文件
s	Socket, 即套接字文件, 用于实现两个进程进行通信

# 操作过程 2:

【操作要求 2】用 shell 命令了解 Linux 文件系统的目录结构

【操作步骤】

执行

```
$ cd /lib
```

```
$ ls -l|more
```

看看/lib 目录的内容,这里都是系统函数.再看看/etc,这里都是系统设置用的配置文件;  
/bin 中是可执行程序; /home 下包括了每个用户主目录。

## 结果 2:

### /lib 目录

```
yanxinyu@ThinkBook16-2022:~$ cd /lib
yanxinyu@ThinkBook16-2022:/lib$ ls -l|more
总用量 652
drwxr-xr-x 2 root root 4096 8月 9 19:49 apg
drwxr-xr-x 2 root root 4096 8月 9 19:49 apparmor
drwxr-xr-x 5 root root 4096 12月 18 15:38 apt
drwxr-xr-x 3 root root 4096 8月 9 19:49 aspell
drwxr-xr-x 2 root root 4096 12月 22 16:16 bfd-plugins
drwxr-xr-x 2 root root 4096 4月 8 2022 binfmt.d
drwxr-xr-x 2 root root 4096 8月 9 19:49 bluetooth
drwxr-xr-x 2 root root 4096 8月 9 19:50 brltty
-rwxr-xr-x 1 root root 1075 12月 8 2021 cnf-update-db
-rwxr-xr-x 1 root root 3565 12月 8 2021 command-not-found
drwxr-xr-x 2 root root 4096 12月 22 16:16 compat-ld
drwxr-xr-x 2 root root 4096 8月 9 19:48 console-setup
lrwxrwxrwx 1 root root 21 12月 18 15:03 cpp -> /etc/alternatives/cpp
drwxr-xr-x 2 root root 4096 8月 9 19:50 crda
drwxr-xr-x 10 root root 4096 8月 9 19:49 cups
drwxr-xr-x 2 root root 4096 12月 18 15:38 dbus-1.0
drwxr-xr-x 3 root root 4096 8月 9 19:50 debug
drwxr-xr-x 3 root root 4096 8月 9 19:48 dpkg
drwxr-xr-x 3 root root 4096 12月 18 15:16 ubiquity
drwxr-xr-x 2 root root 4096 12月 18 15:38 ubuntu-advantage
drwxr-xr-x 2 root root 4096 12月 18 15:38 ubuntu-release-upgrader
drwxr-xr-x 4 root root 4096 12月 18 15:40 udev
drwxr-xr-x 2 root root 4096 8月 9 19:49 udisks2
drwxr-xr-x 2 root root 4096 8月 9 19:49 ufw
drwxr-xr-x 3 root root 4096 8月 9 19:50 unity-settings-daemon-1.0
drwxr-xr-x 2 root root 4096 8月 9 19:50 update-notifier
drwxr-xr-x 3 root root 4096 8月 9 19:48 usrmerge
drwxr-xr-x 2 root root 4096 12月 18 15:38 valgrind
drwxr-xr-x 4 root root 4096 8月 9 19:50 X11
drwxr-xr-x 95 root root 77824 12月 22 16:16 x86_64-linux-gnu
drwxr-xr-x 3 root root 4096 12月 18 15:38 xorg
drwxr-xr-x 2 root root 4096 8月 9 19:50 xserver-xorg-video-intel
yanxinyu@ThinkBook16-2022:/lib$
```

### /etc 目录

```
yanxinyu@ThinkBook16-2022:~$ cd /etc
yanxinyu@ThinkBook16-2022:/etc$ ls -l|more
总用量 1136
drwxr-xr-x 3 root root 4096 8月 9 19:50 acpi
-rw-r--r-- 1 root root 3028 8月 9 19:48 adduser.conf
drwxr-xr-x 3 root root 4096 8月 9 19:49 alsa
drwxr-xr-x 2 root root 4096 12月 22 16:16 alternatives
-rw-r--r-- 1 root root 335 3月 23 2022 anacrontab
-rw-r--r-- 1 root root 433 3月 23 2022 apg.conf
drwxr-xr-x 5 root root 4096 8月 9 19:49 apm
drwxr-xr-x 3 root root 4096 8月 9 19:50 apparmor
drwxr-xr-x 7 root root 4096 12月 18 15:41 apparmor.d
drwxr-xr-x 4 root root 4096 12月 18 15:41 appport
-rw-r--r-- 1 root root 769 2月 23 2022 appstream.conf
drwxr-xr-x 8 root root 4096 12月 18 15:28 apt
drwxr-xr-x 3 root root 4096 8月 9 19:51 avahi
```

```

drwxr-xr-x  2 root root    4096  8月  9 19:50 UPower
-rw-r--r--  1 root root    1523  3月 25 2022 usb_modeswitch.conf
drwxr-xr-x  2 root root    4096  9月  6 2021 usb_modeswitch.d
drwxr-xr-x  2 root root    4096 12月 18 15:41 vim
drwxr-xr-x  4 root root    4096 12月 18 15:28 vmware-tools
lrwxrwxrwx  1 root root      23 12月 18 15:03 vtrgb -> /etc/alternatives/vtrgb
drwxr-xr-x  5 root root    4096  8月  9 19:49 vulkan
-rw-r--r--  1 root root    4942  1月 24 2022 wgetrc
drwxr-xr-x  2 root root    4096  8月  9 19:51 wpa_supplicant
drwxr-xr-x 12 root root    4096  8月  9 19:51 X11
-rw-r--r--  1 root root     681  3月 23 2022 xattr.conf
drwxr-xr-x  6 root root    4096  8月  9 19:49 xdg
drwxr-xr-x  2 root root    4096  8月  9 19:51 xml
-rw-r--r--  1 root root     460 12月  8 2021 zsh_command_not_found
yanxinyu@ThinkBook16-2022:/etc$

```

## /bin 目录

```

yanxinyu@ThinkBook16-2022:~$ cd /bin
yanxinyu@ThinkBook16-2022:/bin$ ls -l|more
总用量 205616
-rwxr-xr-x 1 root root    51632  2月  8 2022 [
-rwxr-xr-x 1 root root    35344  6月 21 2022 aa-enabled
-rwxr-xr-x 1 root root    35344  6月 21 2022 aa-exec
-rwxr-xr-x 1 root root    31248  6月 21 2022 aa-features-abi
-rwxr-xr-x 1 root root    22912  1月 12 2022 aconnect
-rwxr-xr-x 1 root root    19016  1月 25 2022 acpi_listen
-rwxr-xr-x 1 root root    14478  7月 26 19:37 add-apt-repository
-rwxr-xr-x 1 root root    14712  2月 21 2022 addpart

-rwxr-xr-x 1 root root    203768  3月 25 2022 zip
-rwxr-xr-x 1 root root     72088  3月 25 2022 zipcloak
-rwxr-xr-x 1 root root    60065 10月  5 02:16 zipdetails
-rwxr-xr-x 1 root root     2959 10月  8 01:21 zipgrep
-rwxr-xr-x 2 root root   174512 10月  8 01:21 zipinfo
-rwxr-xr-x 1 root root    63896  3月 25 2022 zipnote
-rwxr-xr-x 1 root root    59800  3月 25 2022 zipsplit
-rwxr-xr-x 1 root root    26952  3月 23 2022 zjsdecode
-rwxr-xr-x 1 root root     2206  9月  5 21:33 zless
-rwxr-xr-x 1 root root     1842  9月  5 21:33 zmore
-rwxr-xr-x 1 root root     4577  9月  5 21:33 znew
-rwxr-xr-x 1 root root   875096  3月 25 2022 zstd
lrwxrwxrwx 1 root root      4 12月 18 15:03 zstdcat -> zstd
-rwxr-xr-x 1 root root     3869  3月 25 2022 zstdgrep
-rwxr-xr-x 1 root root      30  3月 25 2022 zstdless
lrwxrwxrwx 1 root root      4 12月 18 15:03 zstdmt -> zstd
yanxinyu@ThinkBook16-2022:/bin$

```

## /home 目录

```

yanxinyu@ThinkBook16-2022:~$ cd /home
yanxinyu@ThinkBook16-2022:/home$ ls -l|more
总用量 4
drwxr-x--- 18 yanxinyu yanxinyu 4096 12月 22 18:01 yanxinyu
yanxinyu@ThinkBook16-2022:/home$

```

## 操作过程 3:

### 【操作要求 3】

用命令分别建立硬链接文件和符号链接文件。通过 `ls -il` 命令所示的 inode、链接

### 【操作步骤】

计数观察它们的区别

找一个其他目录中的文件，如： `/home/zl/mytest.c` 执行

`$ ln /home/zl/mytest.c myt.c` (建立硬链接文件)

`$ ln -s /home/zl/mytest.c myt2.c` (建立符号链接文件)

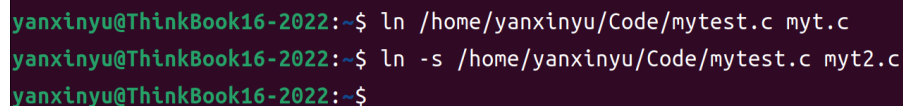
## 结果 3:

`/home/yanxinyu/Code` 路径下有文件 `mytest.c`，其中内容为：



```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]) {
4     printf("Hello, World! \n");
5     return 0;
6 }
```

执行如下命令，建立两种链接。



```
yanxinyu@ThinkBook16-2022:~$ ln /home/yanxinyu/Code/mytest.c myt.c
yanxinyu@ThinkBook16-2022:~$ ln -s /home/yanxinyu/Code/mytest.c myt2.c
yanxinyu@ThinkBook16-2022:~$
```

查看硬链接文件与符号链接文件中的内容，如下所示：



```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]) {
4     printf("Hello, World! \n");
5     return 0;
6 }
```



```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]) {
4     printf("Hello, World! \n");
5     return 0;
6 }
```

可以发现硬链接文件与符号链接文件中的内容均与原来的文件一致。而且经过实验可以发现，只要修改 3 个文件中的任意 1 个文件，都会导致另外 2 个文件的变化。

**思考：** 建立硬链接文件和建立符号链接文件有什么区别，体现在哪里？

**答：** Linux 链接分两种，一种被称为硬链接(Hard Link)，另一种被称为符号链接（Symbolic Link）。默认情况下，ln 命令产生硬链接。

通俗理解，可以把硬链接当成源文件的副本，它和源文件一样的大小但是事实上却不占任何空间。而符号链接可以理解为类似 Windows 系统中的快捷方式。

### 硬链接

- 硬链接指通过索引节点来进行链接。
- 在 Linux 的文件系统中，保存在磁盘分区中的文件不管是什么类型都给它分配一个编号，称为索引节点号(Inode Index)。在 Linux 中，多个文件名指向同一索引节点是存在的。一般这种链接就是硬链接。
- 硬链接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬链接到重要文件，以防止“误删”的功能。其原因如上所述，因为对应该目录的索引节点有一个以上的链接。只删除一个链接并不影响索引节点本身和其它的链接，只有当最后一个链接被删除后，文件的数据块及目录的链接才会被释放。也就是说，文件真正删除的条件是与之相关的所有硬链接文件均被删除。
- 可通过以下命令创建硬链接：

```
ln ExistFile NewFile # 创建硬链接
```

- 硬链接文件有两个限制：不允许给目录创建硬链接、只允许在同一文件系统中的文件之间才能创建链接。
- 对于硬链接文件进行读写和删除操作的时候，结果和符号链接相同。但是如果我们删除硬链接文件的源文件，硬链接文件仍存在，而且保留了原有的内容。

### 软链接

- 另外一种链接称之为符号链接(Symbolic Link)，也叫软链接。软链接文件有类似于 Windows 的快捷方式。它实际上是一个特殊的文件。在符号链接中，文件实际上是一个文本文件，这个文件包含了另一个文件的路径名。
- 可以是任意文件或目录，也可以链接不同文件系统的文件。甚至可以链接不存在的文件，这就产生一般称为“断裂”的问题（现象），还可以不断的循环链接自己。
- 在对符号链接进行读写操作的时候，系统会自动把该操作转换为对源文件的操作。但是删除链接文件时，系统仅仅删除符号链接文件，而不删除源文件本身。
- 可通过以下命令创建软链接：

```
ln -s SourceFile SoftlinkFile # 创建符号链接
```

### 硬链接和符号链接的区别

- 硬链接仅能链接文件，而符号链接可以链接目录
- 硬链接在链接完成后仅和文件内容关联，和之前链接的文件没有任何关系。而符号链接始终和之前链接的文件关联，和文件内容不直接相关。

## 操作过程 4：

### 【操作要求 4】

复习 Unix 或 Linux 文件目录信息 i 节点的概念。编程察看指定文件的 inode 信息

**【提示】** 以下是“获得 inode 信息实验”的例程。将程序稍加修改，进而实现题目要求。

```
#include<stdio.h>
#include<string.h>
#include<time.h>
```



```

#include<unistd.h>
#include<errno.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/sysmacros.h>

#define TIME_STRING_LEN 50

char *time2String(time_t tm, char *buf) {
    struct tm *local;
    local = localtime(&tm);
    strftime(buf, TIME_STRING_LEN, "%c", local);
    return buf;
}

int ShowFileInfo(char *file) {
    struct stat buf;
    char timeBuf[TIME_STRING_LEN];
    if (lstat(file, &buf)) {
        perror("lstat() error");
        return 1;
    }
    printf("\nFile:%s\n", file);
    printf("On device(major/minor):%d %d,inode number:%ld\n",
        major(buf.st_dev), minor(buf.st_dev), buf.st_ino);
    printf("Size:%ld\t Type: %07o\t Permission:%05o\n",
buf.st_size, buf.st_mode & S_IFMT, buf.st_mode & ~(S_IFMT));
    printf("Owner id:%d\t Group id:%d\t Number of hard links:%ld\n",
buf.st_uid, buf.st_gid, buf.st_nlink);
    printf("Last access:%s\n", time2String(buf.st_atime, timeBuf));
    printf("Last modify inode:%s\n\n", time2String(buf.st_atime,
timeBuf));
    return 0;
}

int main(int argc, char *argv[]) {
    int i, ret;
    for (i = 1; i < argc; i++) {
        ret = ShowFileInfo(argv[i]);
        if (argc - i > 1) printf("\n");
    }
    return ret;
}

```

## 结果 4:

```
yanxinyu@ThinkBook16-2022:~/Code$ ./prog5-1.o mytest.c

File:mytest.c
On device(major/minor):8 3,inode number:923754
Size:109          Type: 0100000  Permission:00664
Owner id:1000     Group id:1000   Number of hard links:2
Last access:Fri Dec 23 13:35:39 2022
Last modify inode:Fri Dec 23 13:35:39 2022

yanxinyu@ThinkBook16-2022:~/Code$
```

**思考:** Linux 文件的 inode 是不是很有特色? 找一些这方面的资料, 熟悉文件系统的实现方法, 会让你的水平提升一个台阶的。

**答:** 操作系统的文件数据除了实际内容之外, 通常含有非常多的属性, 例如 Linux 操作系统的文件权限与文件属性。文件系统通常会将这两部分内容分别存放在 inode 和 block 中。

### ① inode 和 block 概述

文件是存储在硬盘上的, 硬盘的最小存储单位叫做扇区 sector, 每个扇区存储 512 字节。操作系统读取硬盘的时候, 不会一个个扇区地读取, 这样效率太低, 而是一次性连续读取多个扇区, 即一次性读取一个块 block。这种由多个扇区组成的块, 是文件存取的最小单位。块的大小, 最常见的是 4KB, 即连续八个 sector 组成一个 block。

文件数据存储在块中, 那么还必须找到一个地方存储文件的元信息, 比如文件的创建者、文件的创建日期、文件的大小等等。这种存储文件元信息的区域就叫做 inode, 中文译名为索引节点, 也叫 i 节点。因此, 一个文件必须占用一个 inode, 但至少占用一个 block。

即元信息→inode, 数据→block。

### ② inode 内容

inode 包含很多的文件元信息, 但不包含文件名, 例如: 字节数、属主 UserID、属组 GroupID、读写执行权限、时间戳等。

而文件名存放在目录当中, 但 Linux 系统内部不使用文件名, 而是使用 inode 号码识别文件。对于系统来说文件名只是 inode 号码便于识别的别称。

### ③ inode 号码

表面上, 用户通过文件名打开文件, 实际上, 系统内部将这个过程分为三步:

- 系统找到这个文件名对应的 inode 号码;
- 通过 inode 号码, 获取 inode 信息;
- 根据 inode 信息, 找到文件数据所在的 block, 并读出数据。

其实系统还要根据 inode 信息, 看用户是否具有访问的权限, 有就指向对应的数据 block, 没有就返回权限拒绝。

在 Linux 中, 可以通过命令 `ls -li` 直接查看文件 i 节点号, 也可以通过 `stat` 查看文件 inode 信息查看 i 节点号。

### ④ inode 大小

inode 也会消耗硬盘空间, 所以格式化的时候, 操作系统自动将硬盘分成两个区域。一个是数据区, 存放文件数据; 另一个是 inode 区, 存放 inode 所包含的信息。每个 inode 的大小, 一般是 128 字节或 256 字节。通常情况下不需要关注单个 inode 的大小, 而是需要重点



关注 inode 总数。inode 总数在格式化的时候就确定了。

在 Linux 中，可以通过命令 `df -i` 查看硬盘分区的 inode 总数和已使用情况。

## 操作过程 5:

### 【操作要求 5】

修改父进程创建子进程的程序，用显示程序段、数据段地址的方法，说明子进程继承父进程的所有资源。再用父进程创建子进程，子进程调用其它程序的方法进一步证明子进程执行其它程序时，程序段发生的变化。

【提示】这个实验可参考例程 3 中“父进程创建子进程，子进程调用其它程序的例”以及下面例程 10“显示程序段、数据段地址的程序”两个程序。设法在子进程运行的程序中显示程序段、数据段地址，以此说明：开始时子进程继承了父进程的资源，一旦子进程运行其它程序，就用该程序替换从父进程处继承来的程序段和数据段。

```
#include<stdio.h>

extern int etext, edata, end;    /*对应文本的第一有效地址、初始化的数据*/
int main() {
    printf("etext:%6p \t edata:%6p \t end:%6p \n", &etext, &edata,
&end);
}
```

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

#define SHW_ADR(ID, I) printf("The id %s \t is at\n", ID, &I);
extern int etext, edata, end;
char *cptr = "Hello World.\n";
char buffer1[25];

int main() {
    void showit(char *);
    int i = 0;
    printf("Adr etext:%8ls\t Adr edata:%8ls Adr end:%8ls\n\n",
&etext, &edata, &end);
    SHW_ADR("main", main);
    SHW_ADR("showit", showit);
    SHW_ADR("cptr", cptr);
    SHW_ADR("buffer1", buffer1);
    SHW_ADR("i", i);
    strcpy(buffer1, "A demonstration\n");
```

```

write(1, buffer1, strlen(buffer1) + 1);
for (; i < 1; ++i)
    showit(cpctr);
}

void showit(char *p) {
    char *buffer2;
    SHW_ADR("buffer2", buffer2);
    if ((buffer2 = (char *) malloc((unsigned) (strlen(p) + 1))) !=
NULL) {
        strcpy(buffer2, p);
        printf("%s", buffer2);
        free(buffer2);
    } else {
        printf("Allocation error.\n");
        exit(1);
    }
}
}

```

## 结果 5:

```

yanxinyu@ThinkBook16-2022:~$ ./prog5-2-1.o
etext:0x55806e1e9199      edata:0x55806e1ec010      end:0x55806e1ec018
yanxinyu@ThinkBook16-2022:~$ ./prog5-2-2.o
Adr etext:                Adr edata:                Adr end:

The id main               is at adr:0x55898d433249
The id showit             is at adr:0x55898d4333f1
The id cpctr              is at adr:0x55898d436010
The id buffer1            is at adr:0x55898d436030
The id i                  is at adr:0x7fff366af434
A demonstration
The id buffer2            is at adr:0x7fff366af410
Hello World.
yanxinyu@ThinkBook16-2022:~$

```

通过实验可以发现，在刚开始时子进程继承了父进程的资源，但当子进程运行其它程序后，就会使用该程序，替换从父进程处继承来的程序段和数据段。

## 操作过程 6:

### 【操作要求 6】

编写一个涉及流文件的程序

- 以只读方式打开一个源文本文件
- 以只读方式打开另一个源文本文件
- 以只写方式打开目标文本文件
- 将两个源文件内容复制到目标文件
- 将目标文件改为指定的属性(其他人只读、文件主可读写)
- 显示目标文件

【提示】这个实验可参考例程 11 中“打开流文件进行行输入输出操作”的例程，当然还得自己加工。将程序再加修改为有两个源文件，一个目标文件，进而实现题目要求。

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
    char s[1024];
    FILE *fp1, *fp2, *fp3;
    if ((fp1 = fopen(argv[1], "r")) == (FILE *) 0) {
        fprintf(stderr, "file1 open error.\n");
        exit(1);
    } else if ((fp2 = fopen(argv[2], "r")) == (FILE *) 0) {
        fprintf(stderr, "file2 open error.\n");
        exit(1);
    } else if ((fp3 = fopen(argv[3], "w")) == (FILE *) 0) {
        fprintf(stderr, "file2 open error.\n");
        exit(1);
    } else {
        while ((fgets(s, 1024, fp1)) != (char *) 0)
            fputs(s, fp3); // 显示缓冲区
        printf("%s", s);
        while ((fgets(s, 1024, fp2)) != (char *) 0)
            fputs(s, fp3); // 显示缓冲区
        printf("%s", s);
    }
    chmod(argv[3], 00755); // 修改文件为指定的属性
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    exit(0);
}
```

### 结果 6:

一个源文本文件 Test6-1

```
"Hello,World!"--Message From File:Test6-1
```

```
"test6-1" 1 line, 42 bytes
```

另一个源文本文件 Test6-2

"From SHU Experiment"--Message From File:Test6-2

```
"test6-2" 1 line, 49 bytes
```

目标文本文件 test6

```
"test6" [readonly] 0 lines, 0 bytes
```

## 运行程序

```
yanxinyu@ThinkBook16-2022:~$ ./prog5-3.o test6-1 test6-2 test6
"Hello,World!"--Message From File:Test6-1
"From SHU Experiment"--Message From File:Test6-2
yanxinyu@ThinkBook16-2022:~$
```

运行程序后的目标文本文件 test6

```
"Hello,World!"--Message From File:Test6-1
"From SHU Experiment"--Message From File:Test6-2

"test6" 2 lines, 91 bytes
```

## 讨论

1. 硬链接文件和符号链接文件, 有什么区别? 系统如何处理的? 举例说明。

**答:** Linux 链接分两种, 一种被称为硬链接(Hard Link), 另一种被称为符号链接 (Symbolic Link)。默认情况下, ln 命令产生硬链接。

通俗理解，可以把硬链接当成源文件的副本，它和源文件一样的大小但是事实上却不占任何空间。而符号链接可以理解类似 Windows 系统中的快捷方式。

## 硬链接和符号链接的区别

- 硬链接仅能链接文件，而符号链接可以链接目录
- 硬链接在链接完成后仅和文件内容关联，和之前链接的文件没有任何关系。而符号链接始终和之前链接的文件关联，和文件内容不直接相关。

## 实验一下

```
$ touch f1                #创建一个测试文件 f1
$ ln f1 f2                #创建 f1 的一个硬连接文件 f2
$ ln -s f1 f3             #创建 f1 的一个符号连接文件 f3
$ ls -li                  # -i 参数显示文件的 inode 节点信息
total 0
7722708 -rw-r--r--  2 yanxinyu yanxinyu 0 12月 30 20:11 f1
7722708 -rw-r--r--  2 yanxinyu yanxinyu 0 12月 30 20:11 f2
7722757 lrwxrwxrwx  1 yanxinyu yanxinyu 2 12月 30 20:11 f3 -> f1
```

从上面的结果中可以看出,硬链接文件 f2 与原文件 f1 的 inode 节点相同,均为 7722708,然而符号链接文件的 inode 节点不同。

```
$ echo "I am f1 file" >>f1
$ cat f1
I am f1 file
$ cat f2
I am f1 file
$ cat f3
I am f1 file
$ rm -f f1
$ cat f2
I am f1 file
cat f3
cat: f3: No such file or directory
```

通过上面的测试可以看出：当删除原始文件 f1 后，硬链接 f2 不受影响，但是符号链接 f1 文件无效

## 2. 从实验 6 的结果可以让我们了解父、子进程之间在资源共享方面是如何处理的？

**答：**Linux 中，内存存储的位置是全局变量，栈区，堆区，以及文件（字符常量区除外）。在数据类型为全局变量、局部变量（栈区）或者动态开辟（堆区）的数据类型时，父子进程之间的数据不共享。对于数据类型为文件时，父子进程之间共享数据，具体而言是共享了文件偏移量。

## 3. 查找资料讨论 Linux 的文件系统有什么特点？它是如何兼容各类文件系统的？

**答：**Linux 系统可以支持十多种文件系统类型包括：JFS、ext、ext2、ext3、ISO9660、XFS、Minx、MSDOS、UMSDOS、VFAT、NTFS、HPFS、NFS、SMB、SysV、PROC 等，Linux 的最重要特征之一就是支持多种文件系统。

Linux 依靠 VFS（Virtual Filesystem Switch，虚拟文件系统转换，也称虚拟文件系统）应对多文件系统，而且支持各种文件系统之间相互访问。VFS 的各种数据结构都是随时建立或删除的，在盘上并不永久存在，只能存放在内存中，也就是说，只有 VFS 是无法工作的，因为它不是真正的文件系统。VFS 向上对应用层提供一个标准的文件操作接口。

## 4. 系统如何管理设备的？怎样体现“与设备无关”的思想方法？

**答：**Linux 把所有外部设备统一当作文件来看待，只要安装它们的驱动程序，任何用户都可以像使用文件一样，操纵、使用这些设备，而不必知道它们的具体存在形式，如此便实现了设备独立性。管理设备与管理文件不同的是，还需要设备控制器、设备与控制器之间的口、缓冲管理等来实现设备管理。将一切都视为文件的最大好处在于，仅需要使用一套接口即可对不同类型的资源进行统一操作。

## 实验体会

通过本次实验中。我了解了硬链接文件和符号链接文件的区别，以及系统是如何处理这两种链接类型的，还了解了父、子进程之间在资源共享方面的处理，使我对文件系统有了更深刻的认识。