第 9 章　目录与文件

9.1　文件操作的权限

所谓权限，指的是文件系统为了进行安全管理需要在对文件操作时进行用户身份认证。合法的用户可以进行操作，而没有权限的用户不能进行文件操作。用 C 程序进行目录与文件操作时，需要设置目录的权限。本节将讲解权限的表示方法。

前面章节讲解的 Linux 命令中，可以用 chmod 命令更改文件的权限。例如下面的命令是对一个文件添加可执行权限。

```
chmod +x a.out
```

在 C 编程中，需要用三个八进制数字来表示文件的权限。例如 777 表示这个用户、同组成员、其他成员对这个文件都有执行、写、读的权限。

三个数字表示的文件权限如下所示。

- 第一个数字表示本用户的权限，相当于 User 的权限。
- 第二个数字表示同组用户的权限，相当于 Group 的权限。
- 第三个数字表示其他用户的权限，相当于 Other 的权限。

数字不同的值代表不同的权限，数字的含义如下所示。

- 4 表示可读权限，相当于 r 权限。
- 2 表示可写权限，相当于 w 权限。
- 1 表示可执行权限，相当于 x 权限。

如果有多种权限，可把几个权限加起来。例如下面不同的权限组合。

- 7 等于 4+2+1，表示这个文件有可读、可写、可执行的权限。
- 5 等于 4+1，表示这个文件有可读、可执行的权限。
- 6 等于 4+2，表示这个文件有可读、可写权限，但是不可以执行。

如果对一个文件设置的权限是 764 表示的权限含义如下所示。

- 7 等于 4+2+1，表示本用户可读、可写、可执行。
- 6 等于 4+2，表示同组用户可读、可写、不可执行。
- 4 含义是其他用户只可读、不可写、不可执行。

chmod 命令也可以使用这些数字作为参数对一个文件更改权限。例如下面的命令就是把一个文件的权限设置为 766。

```
chmod 766 a.out
```

9.2　错误处理与错误号

在进行文件操作时，用户可能会遇到权限不足、找不到文件等错误，这时需要在程序中设置错误捕捉语句并显示错误。错误捕捉和错误输出是应用错误号和 strerror 函数来实现的。

9.2.1　错误定义的理解

Linux 系统已经把所有的错误定义成为不同的错误号和错误常数，程序如果发生了异常，会返回这一个错误的常数。这个常数可以显示为整型数字，也可以用 strerror 函数来显示为已经定义的错误信息。

可以打开包含文件来查看这些错误号的错误信息。在终端中输入下面的命令，打开错误定义文件。

```
vim /usr/include/asm-generic/errno-base.h
```

显示的错误定义代码如下所示。

```
#ifndef _ASM_GENERIC_ERRNO_BASE_H
#define _ASM_GENERIC_ERRNO_BASE_H
#define EPERM        1  /* Operation not permitted */
#define ENOENT       2  /* No such file or directory */
#define ESRCH        3  /* No such process */
#define EINTR        4  /* Interrupted system call */
#define EIO          5  /* I/O error */
#define ENXIO        6  /* No such device or address */
#define E2BIG        7  /* Argument list too long */
#define ENOEXEC     8   /* Exec format error */
#define EBADF       9   /* Bad file number */
#define ECHILD      10  /* No child processes */
#define EAGAIN      11  /* Try again */
#define ENOMEM      12  /* Out of memory */
#define EACCES      13  /* Permission denied */
#define EFAULT      14  /* Bad address */
#define ENOTBLK     15  /* Block device required */
#define EBUSY       16  /* Device or resource busy */
#define EEXIST      17  /* File exists */
#define EXDEV       18  /* Cross-device link */
#define ENODEV      19  /* No such device */
#define ENOTDIR     20  /* Not a directory */
#define EISDIR      21  /* Is a directory */
#define EINVAL      22  /* Invalid argument */
#define ENFILE      23  /* File table overflow */
#define EMFILE      24  /* Too many open files */
#define ENOTTY      25  /* Not a typewriter */
#define ETXTBSY     26  /* Text file busy */
#define EFBIG       27  /* File too large */
#define ENOSPC      28  /* No space left on device */
#define ESPIPE      29  /* Illegal seek */
```

```
#define EROFS       30  /* Read-only file system */
#define EMLINK      31  /* Too many links */
#define EPIPE       32  /* Broken pipe */
#define EDOM        33  /* Math argument out of domain of func */
#define ERANGE      34  /* Math result not representable */
#endif
```

还有一个错误定义文件，可以用下面的命令打开查看这个文件。

```
vim /usr/include/asm-generic/errno.h
```

从这两个文件可知，Linux 系统一共定义 131 种错误常数。用这些错误常数可以返回程序的错误信息。

9.2.2 用错误常数显示错误信息

程序错误返回的错误常数是不容易理解的，需要转换成有效的识别语句。函数 strerror 可以把一个错误常数转换成一个错误提示语句。例如下面的实例是把 ENOENT, EIO, EEXIST 三个错误的常数和错误信息显示出来。

```
#include <stdio.h>
#include <string.h>
#include <errno.h>                          /*包含错误处理头文件。*/
int main(void)
{
   printf("ENOENT:\n");                     /*输出提示。*/
   char *mesg = strerror(ENOENT);           /*将错误号转换成一个字符串。*/
   printf("  Errno  :%d\n",ENOENT);         /*输出错误号。*/
   printf("  Message:%s\n",mesg);           /*输出错误信息。*/

   printf("EIO  :\n");                      /*EIO 错误。*/
   char *mesg1 = strerror(EIO);
   printf("  Errno  :%d\n",EIO);
   printf("  Message:%s\n",mesg1);

   printf("EEXIST   :\n");                  /*文件重名错误。*/
   char *mesg2 = strerror(EEXIST);
   printf("  Errno  :%d\n",EEXIST);
   printf("  Message:%s\n",mesg2);
}
```

这三个错误的含义如下所示。

- ENOENT：没有相关的文件或文件夹。
- EIO：I/O 出现错误。
- EEXIST：文件重名。

> 注意：I/O 指的是计算机的输入和输出，包括一切操作、程序或设备与计算机之间发生的数据流通。最常见的 I/O 设备有打印机、硬盘、键盘和鼠标。

输入下面的命令，编译这个程序。

```
gcc 13.err.1.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
ENOENT:
  Errno  :2
  Message:No such file or directory
EIO    :
  Errno  :5
  Message:Input/output error
EEXIST  :
  Errno  :17
  Message:File exists
```

9.2.3　用错误序号显示错误信息

函数 strerror 可以把一个错误常数返回成一个表示错误信息的字符串。这个函数的使用方法如下所示。

```
char *strerror(int errnum);
```

函数的参数是一个表示错误信息的整型数，返回值是一个字符串。上一节程序中的错误常数实际上也是一个整型数。下面的实例是输出前 15 个序号的错误信息。

```
#include <stdio.h>
#include <string.h>
#include <errno.h>                                  /*包含错误处理头文件。*/
int main(void)
{
   int i;                                           /*定义一个循环变量 i。*/
   for(i=1;i<=15;i++)                               /*循环输出。*/
   {
      printf("Errno:%d  ",i);                       /*输出错误号。*/
      printf("Message:%s\n",strerror(i));           /*输出错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.err.2.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
Errno:1  Message:Operation not permitted
Errno:2  Message:No such file or directory
Errno:3  Message:No such process
Errno:4  Message:Interrupted system call
Errno:5  Message:Input/output error
Errno:6  Message:No such device or address
Errno:7  Message:Argument list too long
Errno:8  Message:Exec format error
Errno:9  Message:Bad file descriptor
Errno:10  Message:No child processes
Errno:11  Message:Resource temporarily unavailable
Errno:12  Message:Cannot allocate memory
Errno:13  Message:Permission denied
Errno:14  Message:Bad address
Errno:15  Message:Block device required
```

9.3 创建与删除目录

在 Linux 系统中，目录就是一个文件夹，文件可以存放在目录中。目录是一种特殊的文件，需要对目录设置权限。本节将讲解 C 程序的目录操作。

9.3.1 创建目录函数 mkdir

函数 mkdir 可以在硬盘中建立一个目录，相当于 mkdir 命令。但与 mkdir 命令不同的是，这里的操作是用 C 语言的函数完成目录创建的。函数的使用方法如下所示。

```
int mkdit(char *pathname,mode_t mode);
```

在参数列表中，pathname 是一个字符型指针，表示需要创建的目录路径，mode 是表示权限的八进制数字。如果目录创建成功，则返回整型数 0，否则返回整型数-1。要使用这个函数需要在程序中包含“sys/types.h”与“sys/stat.h”两个头文件。

在创建目录时，可能有权限、目录名、硬盘空间等问题，可能返回的错误常数如下所示。需要在程序中捕捉这些错误并输出错误信息。

- EPERM：目录中有不合规则的名字。
- EEXIST：参数 pathname 所指的目录已存在。
- EFAULT：pathname 指向了非法的地址。
- EACCESS：权限不足，不允许创建目录。
- ENAMETOOLONG：参数 pathname 太长。
- ENOENT：所指的上级目录不存在。
- ENOTDIR：参数 pathname 不是目录。
- ENOMEM：核心内存不足。
- EROFS：欲创建的目录在只读文件系统内。

- ELOOP：参数 pathname 有多个符合的链接。
- ENOSPC：磁盘空间不足。

下面的实例是在目录/root 下面创建一个 tmp11 目录。程序中设置一个错误号变量，用于捕捉程序发生的异常，如果创建不成功则输出这个错误。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>                          /*包含头文件。*/
#include <errno.h>                             /*包含头文件处理程序的错误。*/
main()
{
   extern int errno;                           /*设置一个错误。*/
   char *path="/root/tmp11";                   /*定义一个字符串表示需要创建的目录。*/

   if(mkdir(path,0766)==0)                     /*创建一个目录。*/
     /*需要注意这里的权限设置参数，第一个 0 表示这里的是八进制数，766 的含义如本章第一节所
     述。*/
     /*如果目录创建成功，则会返回 0，返回值与 0 进行比较。*/
   {
      printf("created the directory %s.\n",path);       /*输出目录创建成功。*/
   }
   else                                        /*如果不成功则输出提示信息。*/
   {
      printf("cant't creat the directory %s.\n",path); /*输出信息。*/
      printf("errno: %d\n",errno);             /*输出错误号。*/
      printf("ERR  : %s\n",strerror(errno));/*输出错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.1.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示，表明已经创建了这个目录。可以在终端中输入“ls /root”命令查看这个目录。

```
created the directory /root/tmp11
```

当再次运行这个程序时，需要创建的目录与上一次创建的目录重名，则不能创建目录，显示的结果如下所示。这时返回了一个错误号，输出了错误信息为“File exists”。

```
cant't creat the directory /root/tmp11.
errno: 17
ERR: File exists
```

9.3.2 删除目录函数 rmdir

rmdir 函数的作用是删除一个目录。该函数的使用方法如下所示。

```
int rmdir(char *pathname);
```

参数 pathname 是需要删除的目录字符串的头指针。如果删除成功则返回一个整型 0，否则返回-1。可以设置一个 errno 来捕获发生的错误。下面是这个函数可能发生的错误。

- EACCESS：权限不足，不允许创建目录。
- EBUSY:系统繁忙，没有删除。
- EFAULT：pathname 指向了非法的地址。
- EINVAL：有这个目录，但是不能删除。
- ELOOP：参数 pathname 有多个符合的链接。
- ENAMETOOLONG：给出的参数太长。
- ENOENT：所指向的上级目录不存在。
- ENOMEM：核心内存不足。
- ENOTDIR：不是一个目录。
- EROFS：指向的目录在一个只读目录里。

下面是使用这个函数来删除一个目录的实例，用来删除上一节创建的目录/root/tmp11。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>                                  /*包含头文件。*/
main()
{
   extern int errno;                                /*设置错误编号。*/
   char *path="/root/tmp11";                        /*设置需要删除的目录。*/

   if(rmdir(path)==0)                               /*删除文件，返回值与 0 比较。*/
   {
      printf("deleted the directory %s.\n",path);       /*显示删除成功。*/
   }
   else                                             /*如果删除不成功。*/
   {
     printf("cant't delete the directory %s.\n",path); /*显示删除错误。*/
     printf("errno: %d\n",errno);                   /*显示错误号。*/
     printf("ERR  : %s\n",strerror(errno));         /*显示错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.2.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示，表明已经删除了这个目录。

```
deleted the directory /root/tmp11.
```

再次运行这个程序，因为没有目录/root/tmp11，所以显示错误信息。程序的运行结果如下所示。

```
cant't delete the directory /root/tmp11.
errno: 2
ERR  : No such file or directory
```

9.4　文件的创建与删除

所谓创建文件，是指在一个目录中建立一个空文件，可供其他程序的写入操作。删除文件指从磁盘中删除无用的文件。本节将讲解文件的建立与删除操作。

9.4.1　创建文件函数 creat

函数 creat 的作用是在目录中建立一个空文件，该函数的使用方法如下所示。

```
int creat(char * pathname, mode_t mode);
```

函数的参数 pathname 表示需要建立文件的文件名和目录名，mode 表示这个文件的权限。文件权限的设置见本章第一节所述。文件创建成功时返回创建文件的编号，否则返回-1。

使用这个函数时，需要在程序的前面包含下面三个头文件。

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
```

如果创建文件不成功，可用 errno 捕获错误编号然后输出。creat 函数可能发生的错误如下所示。

- EEXIST：参数 pathname 所指的文件已存在。
- EACCESS：参数 pathname 所指定的文件不符合所要求测试的权限。
- EROFS：欲打开写入权限的文件存在于只读文件系统内。
- EFAULT：参数 pathname 指针超出可存取的内存空间。
- EINVAL：参数 mode 不正确。
- ENAMETOOLONG：参数 pathname 太长。
- ENOTDIR：参数 pathname 为一目录。
- ENOMEM：核心内存不足。
- ELOOP：参数 pathname 有过多符号链接问题。
- EMFILE：已达到进程可同时打开的文件数上限。

下面的实例是使用 creat 函数在/root 目录下面建立一个文件 tmp.txt。

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

```
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/
main()
{
   extern int errno;                            /*定义错误号。*/
   char *path="/root/tmp.txt";

   if(creat(path,0766)==-1)                     /*用 creat 函数创建一个文件。*/
   {
     printf("cant't create the file %s.\n",path);       /*不能创建文件。*/
     printf("errno: %d\n",errno);               /*显示错误号。*/
     printf("ERR  : %s\n",strerror(errno));     /*显示错误信息。*/
   }
   else                                         /*另一种情况。*/
   {
     printf("created file %s.\n",path);         /*显示创建成功。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.3.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
created file /root/tmp.txt .
```

与创建目录不同的是，当再次运行这个程序时也能创建同名的文件。这时新创建的文件会覆盖以前的文件。

9.4.2 删除文件函数 remove

函数 remove 的作用是删除一个文件。这个函数的使用方法如下所示。

```
int remove(char *pathname);
```

参数 pathname 是一个字符型指针，表示需要删除的目录。文件删除成功则返回 0，否则返回-1。要使用这个函数需要在程序的最前面包含下面的头文件。

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

函数 pathname 在删除文件时，可能发生权限不足或目录不存在的问题。可能产生的错误如下所示，需要在程序中设置 errno 来捕获错误信息。

- EACCESS：权限不足，不允许创建目录。

- EBUSY:系统繁忙，没有删除。
- EFAULT：pathname 指向了非法的地址。
- EINVAL：有这个目录，但是不能删除。
- ELOOP：参数 pathname 有多个符合的链接。
- ENAMETOOLONG：给出的参数太长。
- ENOENT：所指向的上级目录不存在。
- ENOMEM：核心内存不足。
- ENOTDIR：不是一个目录。
- EROFS：指向的目录在一个只读目录里。

下面是用 remove 函数删除上一节创建的文件/root/tmp.txt。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/
main()
{
   extern int errno;                            /*定义一个错误号。*/
   char *path="/root/tmp.txt";                  /*定义要删除的文件名。*/

   if(remove(path)==0)                          /*删除文件。*/
   {
      printf("Deleted file %s.\n",path);        /*显示删除成功。*/
   }
   else                                         /*另一种情况。*/
   {
      printf("cant't delete the file %s.\n",path);      /*显示删除失败。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno)); /*显示错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.4.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。显示已经删除了这个文件。

```
Deleted file /root/tmp.txt.
```

当再次运行这个程序时，显示的结果如下所示。因为需要删除的目录在上一次程序运行

时已经删除，所以找不到要删除的文件。

```
cant't delete the file /root/tmp.txt.
errno: 2
ERR  : No such file or directory
```

9.4.3 建立临时文件函数 mkstemp

所谓临时文件，指的是程序运行时为了存储中间数据而建立的文件。计算机重启时，这些文件会自动删除。如果在程序运行时需要把文件短时间地写到磁盘上，可以使用 mkstemp 函数创建一个临时文件。mkstemp 函数的使用方法如下所示。

```
int mkstemp(char *template);
```

参数 template 表示需要建立临时文件的文件名字符串。文件名字符串中最后六个字符必须是 XXXXXX。mkstemp 函数会以可读写模式和 0600 权限来打开该文件，如果文件不存在则会建立这个文件，返回值是打开文件的编号，如果文件建立不成功则返回-1。

该函数可能发生的错误如下所示。可以用 errno 来捕获错误信息。

- EEXIST：文件同名错误。
- EINVAL：参数 template 字符串最后六个字符非 XXXXXX。

需要注意的是，参数 template 所指的文件名称字符串必须声明为数组，用下面这种声明数组的方法声明。

```
char template[] =”template-XXXXXX”;
```

使用下面这种声明字符串的方法声明的 template 是不能运行的。

```
char *template = “template-XXXXXX”;
```

下面是使用 mkstemp 函数建立文件的实例。

```
#include <stdio.h>
#include <stdlib.h>                                    /*包含头文件。*/
main()
{
   extern int errno;                                   /*设置一个错误号。*/
   char path[]="mytemp-XXXXXX";                        /*定义文件名。*/

   if(mkstemp(path)!=-1)                               /*建立一个临时文件。*/
   {
      printf("created temp file %s.\n",path);          /*显示文件已经建立。*/
   }
   else                                                /*另一种情况。*/
   {
      printf("cant't create temp file %s.\n",path); /*显示不能创建临时文件。*/
      printf("errno: %d\n",errno);                     /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno));           /*显示错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.5.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。显示已经建立了一个临时文件 mytemp-USfaDc。

```
created temp file mytemp-USfaDc.
```

再次运行这个程序，显示的结果如下所示。显示已经建立了一个临时文件 mytemp-hL1La9。系统会自动生成一个不同名的文件名来建立临时文件。

```
created temp file mytemp-hL1La9.
```

在终端中输入“ls”命令，显示的结果如下所示。可以在当前文件夹中查看到这两个临时文件。

```
13.1.c  13.3.c  13.eer.1.o  a.out  mytemp-hL1La9  tmp.c
13.2.c  13.4.c  13.eer.2.c  a.txt  mytemp-USfaDc  tmp.o
```

9.5　文件的打开与关闭

文件的打开指的是从磁盘中找到一个文件，返回一个整型的打开文件顺序编号。打开的文件处于可读、可写状态。文件的关闭指的是释放打开的文件，使文件处于不可读写的状态。

9.5.1　打开文件函数 open

函数 open 的作用是打开一个文件，使文件处于可读写的状态。这个函数的使用方法如下所示。

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

9.5.2 文件打开方式的设置

在上一节所示函数的参数列表中，pathname 表示要打开文件的路径字符串。参数 flags 是系统定义的一些整型常数，表示文件的打开方式。Flags 的可选值如下所示。

- O_RDONLY：以只读方式打开文件。
- O_WRONLY：以只写方式打开文件。
- O_RDWR：以可读写方式打开文件。

上述三种旗标是互斥的，也就是不可同时使用，但可与下列的旗标利用“|”运算符组合。

- O_CREAT：若要打开的文件不存在则自动建立该文件。
- O_EXCL：如果 O_CREAT 已被设置，此指令会去检查文件是否存在。文件若不存在则建立该文件，否则将导致打开文件错误。此外，若 O_CREAT 与 O_EXCL 同时设置时，如果要打开的文件为一个链接，则会打开失败。
- O_NOCTTY：如果要打开的文件为终端机设备时，则不会将该终端机当成进程控制终端机。
- O_TRUNC：若文件存在并且以可写的方式打开时，此标志会清空文件。这样原来的文件内容会丢失。
- O_APPEND：以附加的文件打开文件。当读写文件时会从文件尾开始向后移动，写入的数据会以附加的方式加入到文件后面。
- O_NONBLOCK：以不可阻断的方式打开文件，也就是无论文件有无数据读取或等待操作，都会立即打开文件。
- O_NDELAY：O_NONBLOCK。
- O_SYNC：以同步的方式打开文件，所有的文件操作不写入到缓存。
- O_NOFOLLOW：如果参数 pathname 所指的文件为一符号链接，则会打开失败。
- O_DIRECTORY：如果参数 pathname 所指的文件的目录不存在，则打开文件失败。

9.5.3 打开文件的权限

打开文件时，如果没有这个文件则会自动新建一个文件。在新建文件时需要设置新建文件权限。系统为参数 mode 定义了下面这些常数，可以直接使用这些常数来设置文件的权限。这些权限设置只有在建立新文件时才会有效。

- S_IRWXU：00700 权限，该文件所有者具有可读、可写、可执行的权限。
- S_IRUSR 或 S_IREAD：00400 权限，该文件所有者具有可读取的权限。
- S_IWUSR 或 S_IWRITE：00200 权限，该文件所有者具有可写入的权限。
- S_IXUSR 或 S_IEXEC：00100 权限，该文件所有者具有可执行的权限。
- S_IRWXG：00070 权限，该文件用户组具有可读、可写、可执行的权限。
- S_IRGRP：00040 权限，该文件用户组具有可读的权限。
- S_IWGRP：00020 权限，该文件用户组具有可写入的权限。
- S_IXGRP：00010 权限，该文件用户组具有可执行的权限。
- S_IRWXO：00007 权限，其他用户具有可读、可写、可执行的权限。

- S_IROTH：00004 权限，其他用户具有可读的权限。
- S_IWOTH：00002 权限，其他用户具有可写入的权限。
- S_IXOTH：00001 权限，其他用户具有可执行的权限。

9.5.4　文件打开实例

本节讲解一个文件打开实例，用 open 函数打开一个文件。调用这个函数时，如果正确地打开了这个文件则返回这个文件的打开编号，如果打开失败则返回 0。使用这个函数之前需要在程序的最前面包含下面这些头文件。

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
```

open 函数可能发生下面这些错误。可用 errno 捕获打开文件时发生的错误。

- EEXIST：参数 pathname 所指的文件已存在，却使用了 O_CREAT 和 O_EXCL 旗标。
- EACCESS：参数 pathname 所指的文件没有打开权限。
- EROFS：欲写入权限的文件存在于只读文件系统内。
- EFAULT：参数 pathname 指针超出可存取内存空间。
- EINVAL：参数 mode 不正确。
- ENAMETOOLONG：参数 pathname 太长。
- ENOTDIR：参数 pathname 不在一个目录中。
- ENOMEM：核心内存不足。
- ELOOP：参数 pathname 有过多符号链接问题。
- EIO：I/O 存取错误。

下面的程序是使用 open 函数打开一个文件的实例。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                          /*包含头文件。*/
main()
{
   int fd,fd1;                              /*定义打开文件的编号。*/
   char path[]="/root/txt1.txt";            /*定义需要打开的文件。*/
   extern int errno;                        /*定义错误号。*/
   fd=open(path,O_WRONLY,0766);             /*打开文件，只读，不能自动新建。*/
   if(fd!=-1)                               /*如果返回值不为-1。*/
    {
      printf("opened file %s  .\n",path);   /*显示已经打开文件。*/
    }
   else                                         /*另一种情况。*/
    {
      printf("cant't open file %s.\n",path);    /*显示不能打开文件。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
```

```
        printf("ERR  : %s\n",strerror(errno));     /*显示错误信息。*/
      }

    fd1=open(path,O_WRONLY|O_CREAT,0766);    /*打开文件，如果没有文件则自动新建。*/
    if(fd1!=-1)                              /*如果返回的文件号不为-1。*/
      {
        printf("opened file %s  .\n",path);    /*显示文件新建成功。*/
      }
    else                                     /*另一种情况。*/
      {
        printf("cant't open file %s.\n",path);/*显示不能打开文件。*/
        printf("errno: %d\n",errno);          /*显示错误号。*/
        printf("ERR  : %s\n",strerror(errno)); /*显示错误信息。*/
      }
  }
```

输入下面的命令，编译这个程序。

```
gcc 13.6.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。第一次是以只读的方式打开文件，没有这个文件时显示打开错误。第二次在 Flags 参数中加入了 O_CREAT，表示没有文件时新建这个文件，显示文件打开成功。

```
cant't open file /root/txt1.txt.
errno: 2
ERR  : No such file or directory
opened file /root/txt1.txt  .
```

输入“cd ~”命令进入到用户的根目录，然后输入“ls”命令，可以发现目录中有一个文件 txt1.txt。

9.5.5 关闭文件函数 close

函数 close 的作用是关闭一个已经打开的文件。使用完文件后需要使用 close 函数关闭该文件，这个操作会让数据写回磁盘，并释放该文件所占用的资源。该函数的使用方法如下所示。

```
int close(int fd);
```

参数 fd 是用 open 函数打开文件时返回的打开序号。如果成功关闭文件则返回 0，发生错误则返回-1。在进程结束时，系统会自动关闭已打开的文件，但仍建议在程序中关闭文件，并检查返回值是否正确。在关闭文件时，可能返回 EBADF 错误，表示需要关闭的文件号不存在。

在使用这个函数时，需要在程序的前面包含下面的头文件。

```
#include<unistd.h>
```

下面是一个实例，用 open 函数打开一个文件以后，再用 close 函数关闭这个文件。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/
main()
{
   int fd;                                      /*定义一个文件编号。*/
   char path[]="/root/txt1.txt";                /*定义需要打开的文件名。*/
   extern int errno;                            /*定义一个错误号。*/

   fd=open(path,O_WRONLY|O_CREAT,0766);         /*打开文件，如果没有这个文件则新建。*/
   if(fd!=-1)                                   /*打开成功。*/
   {
      printf("opened file %s  .\n",path);       /*显示打开成功。*/
   }
   else                                         /*打开不成功。*/
   {
      printf("cant't open file %s.\n",path);/*显示打开不成功。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno)); /*显示错误信息。*/
   }

   if(close(fd)==0)                             /*关闭打开的文件。*/
   {
      printf("closed.\n");                      /*显示关闭成功。*/
   }
   else                                         /*另一种情况。*/
   {
      printf("close file %s error.\n",path);/*显示关闭不成功。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno)); /*显示错误信息。*/
   }

    if(close(1156)==0)                          /*关闭一个不存在的文件号。*/
   {
      printf("closed.\n");                      /*如果关闭成功则显示。*/
   }
   else
   {
      printf("close file %s error.\n",path);    /*关闭不成功则显示错误。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno));    /*显示错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.7.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。第一次关闭的是已经打开的一个文件，可以正常关闭。第二次关闭的是一个不存在的文件号，显示没有这个文件。

```
opened file /root/txt1.txt  .
closed.
close file /root/txt1.txt error.
errno: 9
ERR  : Bad file descriptor
```

9.6 文件读写

文件读写指的是从文件中读出信息或将信息写入到文件中。文件读取是使用 read 函数来实现的，文件写入是通过 write 函数来实现的。在进行文件写入的操作时，只是在文件的缓冲区中操作，可能没有立即写入到文件中。需要使用 sync 或 fsynx 函数将缓冲区中的数据写入到文件中。

9.6.1 在文件中写字符串函数 write

函数 write 可以把一个字符串写入到一个已经打开的文件中。这个函数的使用方法如下所示。

```
ssize_t write (int fd,void * buf,size_t count);
```

在参数列表中，fd 是已经打开文件的文件号，buf 是需要写入的字符串，count 是一个整型数，表示需要写入的字符个数。size_t 是一个相当于整型的数据类型，表示需要写入的字节的数目。将字符串写入文件以后，文件的写位置也会随之移动。

如果写入成功，write 函数会返回实际写入的字节数。发生错误发生时则返回-1，可以用 errno 来捕获发生的错误。可能发生的错误如下所示。

- EINTR：此操作被其他操作中断。
- EAGAIN：当前打开文件是不可写的方式打开的，不能写入文件。
- EADF：参数 fd 不是有效的文件编号，或该文件已关闭。

下面是这个函数的使用实例。用 open 函数打开一个文件，将一个字符串写入到这个文件中，然后关闭文件。

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

```
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                                  /*包含头文件。*/
main()
{
   int fd;                                          /*定义打开的文件号。*/
   char path[]="/root/txt1.txt";                    /*定义要写入文件的路径。*/
   char s[]="hello ,Linux.\nI've leart C program for two weeks.\n";
                                                    /*定义要写入的字符串。*/
   extern int errno;                                /*使用错误号。*/

   fd=open(path,O_WRONLY|O_CREAT);                  /*打开文件。*/
   if(fd!=-1)                                       /*打开文件是否成功。*/
   {
      printf("opened file %s  .\n",path);           /*文件打开成功。*/
   }
   else
   {
      printf("cant't open file %s.\n",path);        /*文件打开不成功。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
   }

   write(fd,s,sizeof(s));                           /*将内容写入到文件。*/
   close(fd);                                       /*关闭文件。*/
   printf("Done");                                  /*显示信息。*/
}
```

在函数中，sizeof 可以返回字符串 s 的长度。输入下面的命令，编译这个程序。

```
gcc 13.8.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
opened file /root/txt1.txt .
Done
```

这时在终端中输入下面的命令，查看写入文件的信息。

```
vim  /root/txt1.txt
```

vim 显示的/root/txt1.txt 文件内容如下所示。

```
hello ,Linux.
I've leart C program for two weeks.
```

9.6.2 读取文件函数 read

函数 read 可以从一个打开的文件中读取字符串。这个函数的使用方法如下所示。

```
ssize_t read(int fd,void *buf ,size_t count);
```

在参数列表中，fd 表示已经打开文件的编号。buf 是一个空指针，读取的内容会返回到这个指针指向的字符串。count 表示需要读取字符的个数。返回值表示读取到的字符的个数。如果返回值为 0，表示已经到达文件末尾或文件中没有内容可供读取。在读文件时，文件的读位置会随着读取到的字节移动。

当有错误发生时，返回值为-1，可以用 errno 来捕获错误编号。可能返回的错误如下所示。

- EINTR：此操作被其他操作中断。
- EAGAIN：当前打开文件是不可写的方式打开的，不能写入文件。
- EADF：参数 fd 不是有效的文件编号，或该文件已关闭。

下面是使用 read 函数进行文件读取的实例，可以读取上一节中写入文件的内容，然后显示出读取的字符串和实际读取到的字符数。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/

main()
{
   int fd;                                      /*定义文件号。*/
   char path[]="/root/txt1.txt";                /*定义打开的文件名。*/
   int size;
   char s[100];                                 /*定义一个字符串。*/
   extern int errno;                            /*使用错误号。*/

   fd=open(path,O_RDONLY);                      /*打开文件。*/
   if(fd!=-1)                                   /*打开成功。*/
   {
      printf("opened file %s  .\n",path);
   }
   else
   {
      printf("cant't open file %s.\n",path);    /*打开不成功。*/
      printf("errno: %d\n",errno);              /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno));    /*显示错误信息。*/
   }

   size=read(fd,s,sizeof(s));                   /*读取文件内容。*/
   close(fd);                                   /*关闭文件。*/
   printf("%s\n",s);                            /*输出。*/
   printf("%d\n",size);                         /*输出返回的字节数。*/
}
```

入下面的命令，编译这个程序。

```
gcc 13.9.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。程序显示了文件打开成功和上一节写入的字符串。最后一行显示了实际读取到的字符数。

```
opened file /root/txt1.txt  .
hello ,Linux.
I've leart C program for two weeks.
51
```

9.6.3　文件读写位置的移动

每一个已打开的文件都有一个读写位置。当打开文件时通常读写位置是指向文件开头，若是以附加的方式打开文件，则读写位置会指向文件末尾。read 或 write 函数读或写文件时，读写位置会随读写的字节相应移动。可以用 lseek 函数在文件内容中的位置上面移动，这样就可以在文件中不同的位置进行上读写。这个函数的使用方法如下所示。

```
off_t lseek(int fd,off_t offset ,int whence);
```

在参数列表中，fd 表示用 open 函数打开的文件返回编号，参数 offset 表示根据参数 whence 来移动读写位置的位移数，whence 表示系统定义的常量，可能有下面这些赋值。

- SEEK_SET：参数 offset 即为新的读写位置。
- SEEK_CUR：以目前的读写位置往后增加 offset 个位移量。
- SEEK_END：将读写位置指向文件末尾后再增加 offset 个位移量。

当 whence 值为 SEEK_CUR 或 SEEK_END 时，参数 offet 允许负值的出现。这时表示在当前位置或文件末尾位置上向前移动若干个字节。下面是一些常用的文件位置移动方式。

- lseek（int fd,0,SEEK_SET）：将读写位置移到文件开头。
- lseek（int fd，0,SEEK_END）：将读写位置移到文件结尾。
- lseek（int fd，0,SEEK_CUR）：取得当前的文件位置。

函数调用成功时返回这个文件当前的读写位置，即距文件开头多少个字节。若有错误则返回-1，errno 会存放错误号。这个函数可能产生的错误如下所示。

- EBADF：传入的参数不是一个已经打开的文件。
- EINVAL：给入的 whence 参数不合理。
- EOVERFLOW：给入的移动参数导致文件头指针指向了文件开头以前，产生了溢出错误。

要使用这个函数，需要在程序的前面包含下面的头文件。

- #include<sys/types.h>

- #include<unistd.h>

下面是 lseek 函数的使用实例。打开上一节所使用的文本文件，在文件读取之前，用 lseek 函数移动文件的读写位置。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                                  /*包含头文件。*/
main()
{
   int fd;                                          /*定义文件号。*/
   char path[]="/root/txt1.txt";                    /*定义文件路径。*/
   int size;
   char s[100]="";                                  /*定义一个字符串。*/
   extern int errno;                                /*使用错误号。*/

   fd=open(path,O_RDONLY);                          /*打开文件。*/
   if(fd!=-1)                                       /*打开成功。*/
   {
     printf("opened file %s  .\n",path);
   }
   else
   {
     printf("cant't open file %s.\n",path);         /*打开失败。*/
      printf("errno: %d\n",errno);                  /*显示错误号。*/
      printf("ERR  : %s\n",strerror(errno));        /*显示错误信息。*/
   }

   size=read(fd,s,3);                               /*读取三个字符。*/
   printf("%d  :",size);                            /*输出返回的字符数。*/
   printf("%s\n",s);                                /*输出三个读取的字符。*/

   size=read(fd,s,3) ;                              /*读取后面的三个字符。*/
   printf("%d  :",size);                            /*输出返回的字符数。*/
   printf("%s\n",s);                                /*输出三个读取的字符。*/

   lseek(fd,8,SEEK_SET);                            /*移动到第 8 个字符。*/
   size=read(fd,s,3);                               /*读取后面的三个字符。*/
   printf("%d  :",size);                            /*输出返回的字符数。*/
   printf("%s\n",s);                                /*输出三个读取的字符。*/

   lseek(fd,0,SEEK_SET);                            /*移动到第 0 个字符。*/
   size=read(fd,s,3);                               /*读取后面的三个字符。*/
   printf("%d  :",size);                            /*输出返回的字符数。*/
   printf("%s\n",s);                                /*输出三个读取的字符。*/

   close(fd);                                       /*关闭文件。*/
}
```

输入下面的命令，编译这个程序。

```
gcc 13.10.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。需要注意的是，文件读写时空格也是一个字符，但是输出时看不到这个字符。

```
3  :hel
3  :lo
3  :inu
3  :hel
```

9.6.4　将缓冲区数据写入到磁盘函数 sync

所谓缓冲区，是 Linux 系统对文件的一种处理方式。在对文件进行写操作时，并没有立即把数据写入到磁盘，而是把数据写入到缓冲区中。如果需要把数据立即写入到磁盘，可以用 sync 函数。用这个函数强制写入缓冲区数据的好处是保证数据有同步。这个函数的使用方法如下所示。

```
int sync(void)
```

这个函数会对当前程序打开的所有文件进行处理，将缓冲区中的内容写入到文件。函数没有参数，返回值为 0。这个函数一般不会产生错误。要使用这个函数，需要在程序中包含下面的头文件。

```
#include<unistd.h>
```

下面是这个函数的使用实例。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                                          /*包含头文件。*/

main()
{
   int fd;                                                  /*定义文件号。*/
   char path[]="/root/txt1.txt";                            /*要打开的文件。*/
   char s[]="hello ,Linux.\nI've leart C program for two weeks.\n";    /* 要
写入的字符串。*/
   extern int errno;                                        /*定义错误号。*/

   fd=open(path,O_WRONLY|O_CREAT);                          /*打开文件。*/
```

```
    if(fd!=-1)                                          /*文件打开成功。*/
    {
      printf("opened file %s  .\n",path);
    }
    else
    {
      printf("cant't open file %s.\n",path);            /*文件打开失败。*/
       printf("errno: %d\n",errno);                     /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));           /*显示错误信息。*/
    }
    write(fd,s,sizeof(s));                              /*写入文件。*/
    sync();                                             /*将缓冲区数据写入磁盘。*/
    printf("sync function done. \n");                   /*输出信息。*/
    close(fd);                                          /*关闭文件。*/
}
```

输入下面的命令，编译这个程序。

```
gcc 13.11.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
opened file /root/txt1.txt  .
sync function done.
```

9.6.5 将缓冲区数据写入到磁盘函数 fsync

函数 fsync 的作用是将缓冲区的数据写入到磁盘。与 sync 函数不同的是，这个函数可以指定打开文件的编号，执行以后会返回一个值。这个函数的使用方法如下所示。

```
int fsync(int fd);
```

参数 fd 是 open 函数打开文件时返回的编号。函数如果执行成功则返回 0，否则返回-1。在使用这个函数时，需要在文件前面包含下面的头文件。

```
#include<unistd.h>
```

函数可能发生下面这些错误，可以用 errno 捕获错误。

- EBADF：参数 fd 不是一个正确的文件打开编号或者文件不能写入。
- EIO：发生了 I/O 错误。前面的章节已经讲述过 I / O 错误。
- EROFS 或 EINVAL：fd 是一个特殊的文件，不能够写入内容。

下面程序是 fsync 函数的使用实例。在文件中写入内容以后，用 fsync 函数将文件写入到缓冲区。

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

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                                  /*包含头文件。*/

main()
{
   int fd;                                          /*定义文件号。*/
   char path[]="/root/txt1.txt";                    /*要打开的文件。*/
   char s[]="hello ,Linux.\nI've leart C program for two weeks.\n";    /* 要
写入的字符串。*/
   extern int errno;                                /*定义错误号。*/

   fd=open(path,O_WRONLY|O_CREAT);                  /*打开文件。*/
   if(fd!=-1)                                       /*文件打开成功。*/
   {
      printf("opened file %s  .\n",path);
   }
   else
   {
      printf("cant't open file %s.\n",path);        /*文件打开失败。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
   }

   write(fd,s,sizeof(s));                           /*写入文件。*/
   if(fsync(fd)==0)                                 /*将缓冲区数据写入磁盘。*/
   {
      printf("fsync function done.\n");             /*写入成功。*/
   }
   else
   {
      printf("fsync function failed. \n");          /*写入失败。*/
   }
   close(fd);                                       /*关闭文件。*/
}
```

输入下面的命令，编译这个程序。

```
gcc 9.12.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
opened file /root/txt1.txt  .
```

```
fsync function done.
```

9.7 文件锁定

所谓的文件锁定，指的是以独占的方式打开文件。一个程序打开文件以后，其他的程序不能读取或写入文件。文件锁定有利于文件内容的一致性。本节将讲解文件的锁定问题。

9.7.1 文件锁定的理解

当多个程序同时打开同一个文件时，可能导致文件内容不一致的情况发生。例如一个文件中的数据是一个账户金额。用户打开这个文件读取数据，进行处理以后将结果写入到文件。如果文件没有进行锁定时，可以发生下面这种错误。

（1）假设文件中的数据为 10000。用户 A 打开文件读取这个数据，但是还没有及时的写入。

（2）这时用户 B 读取这个数据，将结果加 10000，然后将结果 20000 写入到这个文件。

（3）用户 A 的处理时间较长，比用户 B 早读入数据，但后写入数据。A 将数据加 1000 得到 11000，然后将数据写入到文件。

这时就发生了一个错误，用户 B 的数据丢失。为了阻止这种错误需要在打开文件时进行文件锁定，正确的做法如下所示。

（1）假设文件中的数据为 10000。用户 A 打开文件读取这个数据，但是还不能及时的写入，于是将文件加一把锁，不允许别的用户访问。

（2）用户 B 访问这个文件时，文件已经加锁，无法访问。于是等待用户 A 完成数据访问。

（3）用户 A 完成数据处理以后，将信息写入到文件，这时解除对文件的锁定。

（4）这时用户 B 可以对文件进行访问。访问的同时对文件添加一个锁定。在解除锁定以前，其他的用户不能访问这个文件。

9.7.2 文件锁定函数 flock

在访问一个文件时，可以用 flock 函数对文件进行锁定，防止其他用户同时访问这个文件发生数据不一致的情况。这个函数的使用方法如下所示。

```
int flock(int fd,int operation);
```

在参数列表中，fd 是 open 函数打开文件时返回的打开序号。operation 是系统定义的一些整型常量，可能的取值和含义如下所示。

- LOCK_SH：建立共享锁定，其他的程序可以同时访问这一个文件。多个程序可同时对同一个文件建立共享锁定。
- LOCK_EX：建立互斥锁定，其他用户不能同时访问这一个文件。一个文件同时只有一个互斥锁定。单一文件不能同时建立共享锁定与互斥锁定。
- LOCK_UN：解除文件锁定状态。
- LOCK_NB：无法建立锁定时，此操作可不被阻断，马上返回进程。通常与 LOCK_SH 或 LOCK_EX 做 OR（|）组合。

当文件锁定成功时会返回 0，否则返回-1。可以用 errno 来捕获发生的错误。这个函数可能发生的错误如下所示。

- EBADF：df 参数不是一个已经打开的文件。
- EINTR：在进行操作时，这个操作被其他的程序或信号所中断。
- EINVAL：给入的参数不合法。
- ENOLCK：核心内存溢出错误。
- EWOULDBLOCK：这个文件已经被其他程序建立互斥锁定。

下面的程序是 flock 函数的使用实例。打开文件以后，对文件建立一个锁定。当其他程序打开这个文件时，则无法打开锁定的文件。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/

main()
{
   int fd,i;                                    /*定义变量。*/
   char path[]="/root/txt1.txt";                /*要访问的文件。*/
   extern int errno;                            /*定义错误号。*/

   fd=open(path,O_WRONLY|O_CREAT);              /*打开文件。*/
   if(fd!=-1)                                   /*打开成功。*/
   {
      printf("opened file %s  .\n",path);
      printf("please input a number to lock the file.\n"); /*输入一个数字。*/
      scanf("%d",&i);                           /*输入。*/
      if(flock(fd,LOCK_EX)==0)                  /*锁定文件。*/
      {
        printf("the file was locked.\n");           /*输出信息。*/
       }
       else
       {
          printf("the file was not locked.\n");     /*文件锁定失败。*/
       }
      printf("please input a number to unlock the file.\n");    /*提示输入。*/
      scanf("%d",&i);                                   /*输入。*/
      if(flock(fd,LOCK_UN)==0)                          /*解除文件锁定。*/
      {
        printf("the file was unlocked.\n");         /*输出解除锁定成功。*/
       }
       else
       {
          printf("the file was not unlocked.\n");   /*输出解除锁定失败。*/
       }
      close(fd);                                    /*关闭文件。*/
```

```
    }
    else
    {
       printf("cant't open file %s.\n",path);          /*不能打开文件的情况。*/
        printf("errno: %d\n",errno);                   /*显示错误号。*/
        printf("ERR  : %s\n",strerror(errno));         /*显示错误信息。*/
    }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.13.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。显示输入一个数字锁定文件。输入一个数字以后，显示文件锁定。这时输入同一个数字可以解除文件锁定。

```
opened file /root/txt1.txt  .
please input a number to lock the file.
```

9.7.3 文件锁定函数 fcntl

函数 fcntl 的作用是对一个文件进行锁定。与 flock 函数不同的是，fcntl 可以设定对文件的某一部分进行锁定。该函数的使用方法如下所示。

```
int fcntl(int fd , int cmd);
int fcntl(int fd,int cmd,long arg);
int fcntl(int fd,int cmd,struct flock *lock);
```

这个函数的参数列表有多种形式。在编译时会自动与参数进行匹配。参数 fd 是 open 函数打开文件时返回的文件编号，cmd 是系统定义的一些整型常量，用来设置文件的打开方式，有下面几种可选方式。

- F_DUPFD：用来查找大于或等于参数 arg 的最小且仍未使用的文件编号，并且复制参数 fd 的文件编号，执行成功则返回新复制的文件编号。
- F_GETFD：取得 close-on-exec 标志。若此标志的参数 arg 的 FD_CLOEXEC 位为 0，代表在调用 exec()相关函数时文件将不会关闭。
- F_SETFD：设置 close-on-exec 旗标。该旗标以参数 arg 的 FD_CLOEXEC 位决定。
- F_GETFL：取得文件描述词状态旗标，此旗标为 open 的参数返回序号。
- F_SETFL： 设置文件描述词状态旗标，参数 arg 为新旗标，但只允许 O_APPEND、O_NONBLOCK 和 O_ASYNC 位的改变，其他位的改变将不受影响。
- F_GETLK：取得文件锁定的状态。
- F_SETLK：设置文件锁定的状态。此时 flcok 结构的 l_type 值必须是 F_RDLCK、

F_WRLCK 或 F_UNLCK。如果无法建立锁定则返回-1，错误代码为 EACCES 或 EAGAIN。

- F_SETLKW 或 F_SETLK：这两个参数的作用相同，无法建立锁定时，此调用会一直等到锁定动作成功为止。

参数 lock 指针为 flock 类型的结构体指针，用来设置锁定文件的一个区域。flock 结构体的定义方式如下所示。

```
struct flock
{
   short int l_type;         /*文件的锁定状态。*/
   short int l_whence;       /*设定 l_start 位置。*/
   off_t l_start;            /*锁定区域的开头位置，从这个地方开始锁定文件的内容。*/
   off_t l_len;              /*锁定文件区域的大小。*/
   pid_t l_pid;              /*锁定文件的进程。*/
};
```

l_type 有下面三种可选的参数。

- F_RDLCK：建立一个供读取用的锁定。
- F_WRLCK：建立一个供写入用的锁定。
- F_UNLCK：删除之前建立的锁定方式。

l_whence 有下面的三种可选设置方式。

- SEEK_SET：以文件开头为锁定的起始位置。
- SEEK_CUR：以文件当前的读写位置为锁定的起始位置。
- SEEK_END：以文件结尾为锁定的起始位置。

fcntl 函数执行成功则返回 0，失败则返回-1。这个函数可能发生下面的错误，可以用 errno 来捕获发生的错误。

- EACCES 或 EAGAIN：文件已经被锁定，没有权限再进行锁定操作。
- EAGAIN：文件已经被另外一个进程占用。
- EBADF：df 参数无效或文件已经关闭。
- EDEADLK：这个文件被死锁。
- EFAULT：文件处于只读的分区内。
- EINTR：操作被其他的程序或信号中断。
- EINVAL：函数所给的参数不合法。
- EMFILE：已经超过了最多可以打开的文件数。
- ENOLCK：已经建立了太多的锁定方式。
- EPERM：在以追加方式打开的文件上错误的进行锁定。

在使用这个函数时，需要在程序的最前面包含下面的头文件。

```
#include <unistd.h>
#include <fcntl.h>
```

9.7.4 文件锁定函数 fcntl 使用实例

本节讲解 fcntl 函数的使用实例。打开一个文件后，用 fcntl 函数来锁定文件的一部分字节。在程序中需要定义一个 flock 类型的结构体，然后设置这个结构体各个成员的变量。用这个结构体的指针作为 fcntl 函数的参数。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                                  /*包含头文件。*/

main()
{
   int fd;                                          /*定义打开的文件名。*/
   char path[]="/root/txt1.txt";                    /*定义要打开的文件名字符串。*/
   struct flock fl;                                 /*定义一个 flock 结构体。*/
   char s[]="hello ,Linux.\nI've leart C program for two weeks.\n";
   extern int errno;                                /*定义一个错误号。*/

   fd=open(path,O_WRONLY|O_CREAT);                  /*打开文件。*/
   if(fd!=-1)                                       /*文件打开成功。*/
   {
      printf("opened file %s  .\n",path);           /*输出文件打开成功。*/
   }
   else
   {
      printf("cant't open file %s.\n",path);        /*文件打开失败。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
   }

   fl.l_type=F_RDLCK;                               /*构造 fl 结构体的内容。*/
   fl.l_whence=SEEK_SET;                            /*文件位置方式。*/
   fl.l_start=2;                                    /*从第二个字节开始。*/
   fl.l_len=10;                                     /*一共锁定 10 个字节。*/
   fl.l_pid=15;                                     /*进程号。*/

   if(fcntl(fd,F_SETLKW,&fl)==0)                    /*锁定文件的这个区域。*/
   {
      printf("some string of the file was locked.\n");  /*显示锁定成功。*/
   }
   else
   {
       printf("locked error.\n");                   /*锁定不成功则显示错误。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
    }
    close(fd);                                      /*关闭文件。*/
```

```
}
```

输入下面的命令，编译这个程序。

```
gcc 13.14.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示，表示文件从第二个字节开始以后的 10 个字节已经被锁定。

```
some string of the file was locked.
```

9.8　文件的移动与复制

文件的移动指的是把文件中一个目录中转移到另一个目录中，C 程序提供了方便的文件移动函数。文件的复制指的是将文件作一个备份，C 程序没有提供文件复制函数。需要新建一个文件，从原文件中读取内容写入到新文件中。本节将讲解文件的移动与复制操作。

9.8.1　文件的移动函数 rename

在 Linux 系统中，移动文件有两种方式。一种方式是在同一个分区中移动文件，这种文件移动方式相当于把文件进行重命名。另一种方式是在不同分区之间移动文件。本节只讲前一种文件移动方式。

在同一个分区中移动文件可以用 rename 函数。该函数的使用方式如下所示。

```
int rename(char *oldpath, char *newpath);
```

在参数列表中，oldpath 表示原文件的路径，newpath 表示文件的新路径。rename 函数可以把文件从原路径移动到新路径中。如果文件移动成功将返回 0，不成功返回-1。文件移运不成功时，可能产生下面这些错误。可以用 errno 捕获程序中的错误。

- EACCES：文件的目录不可写或没有可写权限。
- EBUSY：文件被其他程序占用，处于繁忙状态。
- EFAULT：新文件或旧文件处于不可访问的目录中。
- EINVAL：文件名指向的目录不存在。
- EISDIR：旧文件是一个目录，或新文件是一个目录。
- ELOOP：新文件或旧文件有太多的文件或链接相匹配。
- EMLINK：文件目录中的文件已经到了最大数目录。
- ENAMETOOLONG：文件名太长。
- ENOENT：旧文件或新文件不存在。
- ENOMEM：内核内存不足。
- ENOSPC：磁盘的空间不足。

- ENOTDIR：新文件或旧文件指向的目录不存在。
- EROFS：新文件处于只读分区内。
- EXDEV：新文件和旧文件不是在同一个类型的文件分区内。

使用这个函数时，需要在程序的最前面包含下面的头文件。

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

9.8.2 rename 函数使用实例

本节讲述一个实例，使用 rename 函数移动一个文件。将/root/a.txt 文件移到到/tmp 目录中，文件名为 b.txt。在使用这个函数时，需要注意对错误信息的处理。

```
#include <stdio.h>
#include <errno.h>                                  /*包含头文件。*/

main()
{
   char path[]="/root/a.txt";                       /*原文件。*/
   char path1[]="/root/a11.txt";                    /*目录文件。*/
   char newpath[]="/tmp/b.txt";                     /*一个不存在的文件。*/
   extern int errno;                                /*错误号。*/
   if(rename(path,newpath)==0)                      /*移动文件。*/
   {
      printf("the file %s was moved to %s .\n",path,newpath);   /*移动成功。*/
   }
   else
   {
      printf("can't move the file %s .\n",path);    /*显示移动失败。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
   }

   if(rename(path1,newpath)==0)                     /*移动一个不存在的文件。*/
   {
      printf("the file %s was moved to %s .\n",path1,newpath); /*显示结果。*/
   }
   else
   {
      printf("can't move the file %s .\n",path1);   /*显示移动不成功能。*/
       printf("errno: %d\n",errno);                 /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));       /*显示错误信息。*/
   }
}
```

输入下面的命令，编译这个程序。

```
gcc 13.15.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。第一次是移动一个已经存在的文件，显示文件移动成功。第二次是移动一个不存在的文件，显示文件移动失败。显示的错误信息是没有这个文件。

```
the file /root/a.txt was moved to /tmp/b.txt .
can't move the file /root/a11.txt .
errno: 2
ERR  : No such file or directory
```

在终端中输入下面的命令，查看/tmp 文件夹下有没有移动的文件。

```
ls /tmp/b.txt
```

显示的结果如下所示。表明/tmp 文件夹下面有程序中移动的文件。

```
/tmp/b.txt
```

9.8.3　文件复制实例

在 C 程序中，没有直接复制一个文件的函数。如果需要复制一个文件，可以分别打开源文件和目标文件。依次从源文件中读取一定长度的内容，然后写入到目标文件中。下面的程序是使用这种方法进行文件复制的实例。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>                              /*包含头文件。*/

main()
{
   int fd,fd2,size,i=1;
   char path[]="/root/s.txt";                   /*要打开的文件。*/
   char newpath[]="/root/a2.txt";               /*复制的目录文件。*/
   char buf[100];                               /*定义一个字符串。*/
   extern int errno;                            /*定义一个错误号。*/

   fd=open(path,O_RDONLY);                      /*打开源文件。*/
   fd2=open(newpath,O_WRONLY|O_CREAT);          /*打开目标文件。*/
   if(fd!=-1)
   {
      printf("opened file %s  .\n",path);       /*源文件打开成功。*/
   }
   else
   {
      printf("cant't open file %s.\n",path);    /*源文件打开失败。*/
       printf("errno: %d\n",errno);             /*显示错误号。*/
       printf("ERR  : %s\n",strerror(errno));   /*显示错误信息。*/
```

```
    }
    for(;i!=0;)
    {
        i=read(fd,buf,sizeof(buf));          /*从源文件中读取内容。*/
        printf("%d",i);                      /*输出读取的字节个数。*/
        printf("%s",buf);                    /*输出内容。*/
        if(i==-1)                            /*到达末尾则结束。*/
        {
           break;                            /*结束循环。*/
        }
        else
        {
           write(fd2,buf,sizeof(buf));       /*将读取的内容写入到目标文件中。*/
        }
    }
    close(fd);                               /*关闭文件。*/
    close(fd2);                              /*关闭文件。*/
}
```

输入下面的命令，编译这个程序。

```
gcc 13.15.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。显示文件已经复制。

```
opened file /root/s.txt  .
file was copied.
```

在终端中输入下面的命令，可以查看到已经复制的文件。

```
ls /root
```

9.9 文件实例：电话本程序

在程序中处理的数据需要记录到文件中才能够长期保存。本章所讲的文件操作可以用来进行各种文件的读写操作。本节将讲述一个文件使用实例，设计一个电话号码管理程序，电话号码的内容使用文件读写函数记录到文件中。在本章的学习中，需要重点理解文件的读写操作。

9.9.1 程序功能分析

在设计一个程序之前，需要对程序的各种功能进行规划，安排程序的各个模块和实现方式。本节将对这个程序的需要和实现方法进行大体分析。

一个电话号码管理程序需要对电话号码进行添加、删除、保存、打开等操作。这个程序需要完成下面这些功能。

（1）进行程序以后，需要有一个选择菜单。

（2）需要添加电话号码，实现数据的输入。

（3）需要删除电话号码，实现数据的管理。

（4）需要进行数据列表，显示所有的电话号码信息。

（5）需要根据一个姓名进行查找，查出这个用户的电话。

（6）要有文件写入功能，把信息保存到文件上。

（7）要有文件读取功能，从文件中读取以前保存的数据。

9.9.2　程序的函数

这个程序实现了复杂的功能，需要把这些功能写成不同的函数模块。这样可以把一个复杂的程序拆分成多个独立的简单模块，然后用主函数把这些程序组织到一起。这个程序需要编写下面这些模块。

- 菜单函数，完成菜单的显示和选择。

```
int menu();
```

- 显示电话函数，参数是一个结构体，显示结构体的信息。

```
void shownum(struct telnum t)
```

- 添加电话函数，完成一个电话号码的添加，返回一个结构体。

```
struct telnum addnum()
```

- 查找函数，用户输入一个姓名，查找到这个用户的电话。

```
void selectbyname()
```

- 删除电话号码函数，用户输入一个姓名，删除这个用户的电话号码。

```
void delenum()
```

- 保存信息功能，将所有电话号码保存到文件上。

```
void savetofile()
```

导入电话号码，从文件中读取以前的文件信息。

```
void loadfromfile()
```

主函数，完成各个函数的调用。

```
main()
```

9.9.3　包含文件

在这个程序中，需要进行字符串比较、字符串复制、错误处理、文件读写等操作。这些操作需要包含相应的头文件。程序的第一部分是包含需要使用的头文件，代码如下所示。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>                                  /*包含相关的头文件。*/
```

9.9.4 数据的定义

程序中的电话号码和姓名之间存在着一对一的对应关系，需要定义一个结构体来体现出这种对应关系。电话号码和计数变量是全局变量，程序中所有的函数需要访问这些数据。程序的这一部分需要定义一个结构体，定义程序的全局变量，代码如下所示。

```
struct telnum
{
   char name[7];                                    /*姓名。*/
   char tel[13];                                    /*电话号码。*/
};                                                  /*定义保存一个电话号码的结构体。*/
struct telnum num[100];                             /*全局变量，一个结构体数组。*/
int i;                                              /*全局变量，当前记录的条数。*/
```

9.9.5 菜单函数

程序中有很多功能，需要把这些功能用一个菜单供用户选择。每一个菜单有一个编号，用户从键盘输入相对应的编号。函数对输入的编号进行判断，如果输入有效则返回这个选项，如果输入的选项为 0 则退出这个程序，如果输入的选项无效则显示错误要求用户重新选择。这个函数的运行流程如图 9-1 所示。

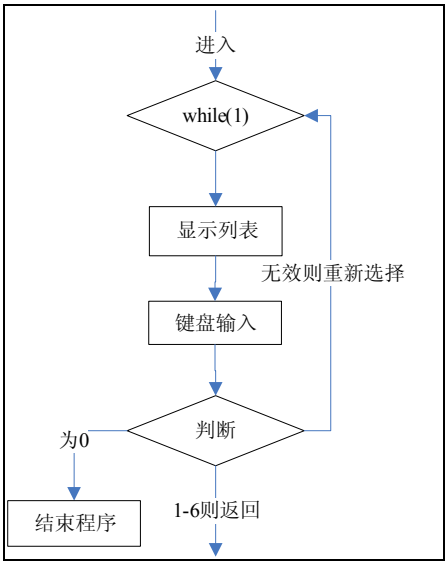

图 9-1 菜单选择的流程图

根据这个菜单选择的流程图编写的程序代码如下所示。

```
int menu()
{
   int i;                                          /*选择的数字。*/
   i=0;                                            /*赋初值为 0。*/
   while(1)
   {
      printf("Please select a menu:\n");           /*提示输入。*/
      printf("      1: add a number.\n");          /*选项。*/
      printf("      2: all the number.\n");
      printf("      3: select a number by name.\n");
      printf("      4: delete a number .\n");
      printf("      5: save to file .\n");
      printf("      6: load numbers from file .\n");
      printf("      0: exit .\n");
      scanf("%d",&i);                              /*输入信息。*/
      if(i==0)                                     /*如果为 0 则退出。*/
      {
         printf("Byebye.\n");
         exit(1);                                  /*退出程序。*/
      }
      if(i<1 || i>6)                               /*不在正确的范围则显示错误。*/
      {
         printf("error.\n");
         continue;                                     /*下次循环。*/
      }
      else
      {
         return(i);                                    /*正常范围则返回这个值。*/
      }
   }
}
```

9.9.6　显示电话信息函数

这个函数的作用是显示一条电话号码的信息，参数是一个电话信息结构体。函数访问这个结构体的成员变量，输出这些信息。完成数据的输出以后，自动结束程序，没有返回值。函数的代码如下所示。

```
void shownum(struct telnum t)
{
    printf("Name   :%s\n",t.name);                 /*显示姓名。*/
    printf("    tel:%s\n",t.tel);                  /*显示电话。*/
}
```

9.9.7　添加电话号码函数

这个函数的作用是提示用户输入电话和姓名信息，然后返回一个电话号码结构体。程序中需要注意的是，这里使用 strcpy 函数将输入的字符串信息复制到结构体的成员中。这个函

数没有参数，返回值是一个结构体。

```
struct telnum addnum()
{
    struct telnum numtmp;                          /*定义一个电话号码结构体变量。*/
    char na[7],tel[13];                            /*定义两个数组。*/
    printf("add a telephone number:\n");           /*提示信息。*/
    printf("please input the name:\n");
    scanf("%s",na);                                /*输入姓名。*/
    printf("please input the num:\n");             /*输入电话号码。*/
    scanf("%s",tel);
    strcpy(numtmp.name,na);                        /*变量复制到结构体的成员上。*/
    strcpy(numtmp.tel,tel);
    return(numtmp);                                /*返回这个结构体。*/
}
```

9.9.8 按姓名查找函数

这个函数的作用是提示用户输入一个姓名并查找此用户的电话号码。函数用 for 循环语句访问结构体数组中的每一个姓名。将每个姓名与当前输入的姓名做比较，如果相同则输出这条记录。程序中使用了一个计数器，如果没有输出任何记录则输出错误提示。

```
void selectbyname()
{
    char na[20];                                   /*定义一个字符串。*/
    int n,j;                                       /*计数变量。*/
    n=0;
    printf("select a number by name:\n");          /*输入姓名。*/
    printf("please input a name :\n");
    scanf("%s",na);                                /*输入。*/
    for(j=0;j<i;j++)                               /*循环访问结构体。*/
    {
        if(strcmp(num[j].name,na)==0)              /*比较输入的变量与结构体中姓名是否相
                                                   同。*/
        {
            shownum(num[j]);                       /*相同则输出。*/
            n++;                                   /*计数加 1。*/
        }
    }
    if(n==0)
    {
        printf("no such a name");                  /*如果 n 为 0 就显示没有这一条记录。*/
    }
}
```

9.9.9 删除电话号码函数

这个函数的作用是从电话号码数组中删除一条电话号码信息。实例的方法是让用户输入的姓名与数组中的每一个姓名进行比较。如果有一条记录中的姓名与输入姓名相同，则将数

组中这条记录后面的所有记录向前移动一个位置。这样便覆盖了需要删除的电话号码。同时，需要将电话号码总数减 1。程序的代码如下所示。

```
void delenum()
{
    char na[20];                                /*定义一个字符串。*/
    int j,n;                                    /*定义计数变量。*/
    n=0;                                        /*计数变量赋初值为 0。*/
    printf("delete a num by name:\n");          /*提示。*/
    printf("please input a name :\n");          /*提示。*/
    scanf("%s",na);                             /*输入一个姓名。*/
    for(j=0;j<i;j++)                            /*循环访问结构体数组中的每一个姓名。*/
    {
        if(strcmp(num[j].name,na)==0)           /*比较输入的姓名与结构体数组中的每一
                                                  个姓名。*/
        {
            n++;                                /*相同则计数加 1。*/
            for(;j<i;j++)                       /*把后面的变量向前移动。*/
           {
              num[j]=num[j+1];
           }
           i--;                                 /*总数减 1。*/
            break;                              /*结束循环。*/
        }
    }
    if(n==0)                                    /*如果计数为 0 表示没有这个姓名。*/
    {
        printf("no such a name");
    }
}
```

9.9.10　保存到文件函数

这个函数的作用是将用户输入的所有电话号码保存到一个文本文件中，函数没有参数和返回值。需要保存的内容都已保存到全局变量中，这个函数可以通过访问全局变量的内容将需要保存的内容保存到一个文本文件中。函数用“open”函数打开一个文件，如果没有文件会自动创建这个文件，然后使用 for 循环访问全局变量数组中的每一个结构体成员，将这些信息依次写入到文件中。函数的代码如下所示。

```
void savetofile()
{
   int j,fd;                                    /*定义变量。*/
   char file[]="/root/tel.txt";                 /*定义文件名。*/
   extern int errno;                            /*设置错误号。*/
   fd=open(file,O_WRONLY|O_CREAT);              /*打开文件。*/
   if(fd!=-1)                                   /*打开正常。*/
   {
     printf("opened file %s  .\n",file);        /*显示正常。*/
   }
```

```
    else                                          /*不能打开文件就显示错误。*/
    {
       printf("cant't open file %s.\n",file);
        printf("errno: %d\n",errno);
        printf("ERR  : %s\n",strerror(errno));
    }
    for(j=0;j<i;j++)                              /*循环访问结构体数组，写入变量。*/
    {
        printf(" %d %s\n",j,num[j].name);         /*显示姓名。*/
       write(fd,num[j].name,7);                   /*保存姓名。*/
       write(fd,num[j].tel,13);                   /*保存电话。*/
    }
    printf("saved.\n");                           /*输出信息。*/
    close(fd);                                    /*关闭文件。*/
}
```

9.9.11 从文件导入信息函数

本函数的作用是从以前保存的文件中读取电话号码和姓名信息，并将这些信息保存到全局变量的数组中。在导入信息时，依次读取若干个字符。如果读取的字符个数不为 0，则在结构体数组中添加一条记录。函数的代码如下所示。

```
void loadfromfile()
{
    int j=0,fd,t;                                 /*定义变量。*/
    i=0;
    char na[7];                                   /*定义姓名。*/
    char tel[13];                                 /*定义电话。*/
    char file[]="/root/tel.txt";                  /*定义文件名。*/
    extern int errno;                             /*设置错误号。*/
    fd=open(file,O_WRONLY|O_CREAT);               /*打开文件。*/
    if(fd!=-1)                                    /*文件打开成功。*/
    {
      printf("opened file %s  .\n",file);
    }
    else                                          /*文件打开失败。*/
    {
      printf("cant't open file %s.\n",file);
       printf("errno: %d\n",errno);
       printf("ERR  : %s\n",strerror(errno));
    }
    while((t=read(fd,na,7))!=0&&t!=-1)             /*读取文件。*/
    {
        strcpy(num[i].name,na);                   /*复制字符串到结构体中。*/
        read(fd,tel,13);                          /*读取电话。*/
        strcpy(num[i].tel,tel);                   /*复制字符串到结构体中。*/
        i++;                                      /*计数加 1。*/
     }
    close(fd);                                    /*关闭文件。*/
```

```
}
```

9.9.12　主函数

主函数的作用是将上面编写的这些函数用一定的关系组织起来，统一完成一个程序功能。在程序开始时，进入一个 while 循环，在循环中反复调用 menu 函数来进行功能选择，然后根据返回的数值来调用相关的子函数。主函数的流程可用图 9-2 来表示。

根据图中的程序运行流程编写的主函数如下所示。

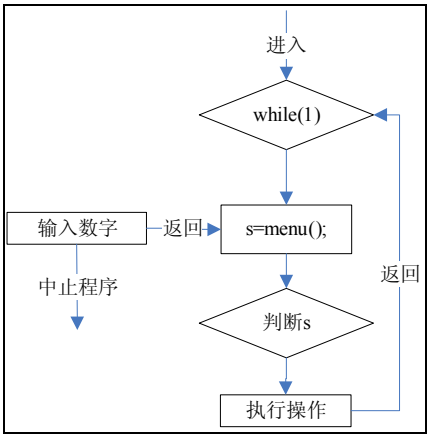

图 9-2　程序主函数的流程图

```
main()
{
    int s,j;                                    /*定义变量。*/
    printf("- - - - Telphone Notebook. - - - \n"); /*输出提示。*/

    while(1)                                    /*进入一个死循环。*/
    {
       s=menu();                                /*显示菜单。*/
       if(s==1)                                 /*分别判断 s 的值，调用不同的函数。*/
      {
          num[i]=addnum();                      /*添加一个电话号码。*/
          i++;                                  /*总数加 1。*/
       }
       if(s==2)                                 /*显示所有记录。*/
       {
          for(j=0;j<i;j++)                      /*循环。*/
          {
             shownum(num[j]);                   /*显示所有记录，结构体作为参数。*/
           }
        }
        if(s==3)                                /*按姓名查找。*/
        {
          selectbyname();
        }
        if(s==4)                                /*删除一个记录。*/
        {
          delenum();
        }
       if(s==5)                                 /*保存到文件。*/
       {
          savetofile();
        }
        if(s==6)                                /*从文件中导入。*/
        {
```

```
            loadfromfile();
          }
      }
  }
```

9.9.13　程序的运行

输入下面的命令，编译这个程序。

```
gcc 13.2.c
```

输入下面的命令，对编译的程序添加可执行权限。

```
chmod +x a.out
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。程序最开始显示一个选择菜单，要求用户选择一个功能。

```
- - - - Telphone Notebook. - - -
Please select a menu:
      1: add a number.
      2: all the number.
      3: select a number by name.
      4: delete a number .
      5: save to file .
      6: load numbers from file .
      0: exit .
```

用户输入“1”，然后按“Enter”键，提示用户输入姓名和电话。用同样的方法输入三个姓名和电话。这时选择“2”，显示所有的电话，结果如下所示。

```
Name   :jim
   tel:65455676
Name   :Lily
   tel:65746367
Name   :Lucy
   tel:98347834
```

用户选择“4”，然后输入一个姓名，可以删除已经添加的姓名和电话。如果没有这个姓名，程序会显示没有这条记录。

用户选择“5”，程序会把输入的信息保存到文件“/root/tel.txt”中。这时输入下面的命令，查看保存的信息。

```
vim /root/tel.txt
```

VIM 显示的文本信息如下所示。

```
jim65455676Lily65746367Lucy98347834
```

当程序再次启动时，输入“6”，可以从这个文件导入上次保存的信息。输入“0”可以退

出这个程序。

9.10 小结

本章讲解了目录和文件的基本操作。其中需要掌握的知识包括目录的创建与删除、文件的创建与删除、文件的移动与复制、文件的锁定与权限、文件的缓冲区操作和文件的读写。其中文件的读写是本章的难点，需要做大量的编程练习。最后的综合实例讲解了文件在程序中的运用方法，难点是将程序处理的信息保存到文件。文件内容的具体操作，将在下一章中讲到。