# 文件IO

#### I/O比较模型

|  |  |  |
| --- | --- | --- |
| I/O模型 | 文件IO | 标准IO |
| 缓冲方式 | 非缓冲IO | 缓冲IO |
| 操作对象 | 文件描述符 | 流(FILE\*) |
| 打开 | open() | fopen()/fropen()/fdopen() |
| 读 | read() | fread()/fgetc()/fgets() |
| 写 | write() | fwrite()/fputc()/fputs() |
| 定位 | lseek() | fseek()/ftell()/rewind()/fsetpos()/fgetpos() |
| 关闭 | close() | fclose() |

主要就是三种不同：

1）缓冲方式不同：标准io有缓冲，文件io无缓冲（这里的缓冲指的是是否有C库里面的缓冲，文件io属于系统调用里的，自然不会有这种缓冲，但在内核里面还是有缓冲的）

2）操作对象不同：标准io操作对象是流，也就是文件指针FILE\*，文件io操作对象是文件描述符，后面会具体介绍

3）使用的操作函数不同：见上表

#### 文件描述符

##### 什么是文件描述符

文件描述符是一个顺序分配的非负整数

非负整数：文件描述符从0开始往上递增，在Linux系统中每一个进程默认最多打开1024个文件描述符

顺序分配：在分配时总是返回最小可以用的文件描述符

注意：每个进程在启动的时候会默认自动打开3个文件描述符（0、1、2）

##### 每个程序在运行时会默认自动打开三个标准终端

标准输入（行缓冲）、标准输出（行缓冲）、标准错误输出（无缓冲）

|  |  |  |  |
| --- | --- | --- | --- |
| 描述 | 文件描述符 | 文件描述符对应的宏 | 文件指针（FILE\*） |
| 标准输入 | 0 | STDIN\_FILENO | stdin |
| 标准输出 | 1 | STDOUT\_FILENO | stdout |
| 标准错误输出 | 2 | STDERR\_FILENO | stderr |

##### 文件描述符指代什么

文件描述符（fd）对应的是打开的文件的描述信息

#### 系统调用和库函数

##### 系统调用

在man手册的第二部分，是用户空间进入到内核空间的一个入口

屏蔽硬件差异，让应用层程序不用关心底层是硬件编程

提高系统的安全性与可移植性

在整个Linux中有两百多个不到三百个系统调用函数

##### 库函数

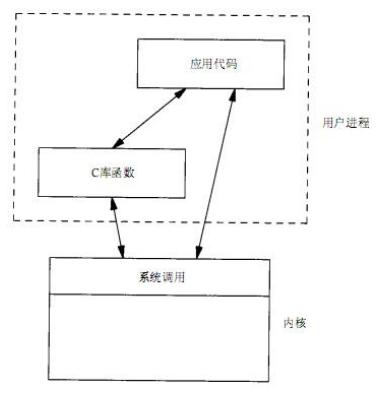
在man手册第三部分

库函数是为了实现某些特定功能而封装起来的一些代码合集

在库函数里可以调用系统调用，也可以不调用（比如C库，math库，pthread线程库等）

虽然说库函数和系统调用在底层实现上有很大的差别，但是对于用户编程来讲，其实没什么差别，只不过是调用的函数不同罢了。

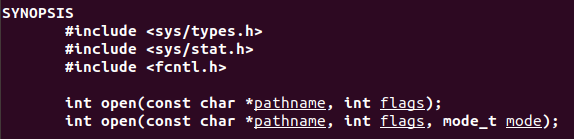
##### 3）层级图



#### 文件IO操作函数

##### open()函数

函数原型：



我们会发现有两种函数写法，但却并不是像printf那样是个变参函数。

函数描述：

open()系统调用打开由pathname指定的文件。如果指定的文件不存在，open()可以有选 择地创建它(如果在flags中指定了O\_CREAT)。

参数描述：

pathname:指定文件的路径

flags:

|  |  |  |
| --- | --- | --- |
| O\_RDONLY | 只读方式打开文件 | 此三个参数互斥 |
| O\_WRONLY | 只写方式打开文件 |
| O\_RDWR | 读写方式打开文件 |
| O\_CREAT | 如果文件不存在就创建文件 |  |
| O\_EXCL | 如果使用O\_CREAT时文件存在，则可返回一个错误消息，这一参数可以判断文件是否存在 |  |
| O\_NOCITY | 使用本参数时若文件为终端，则终端不可作为调用open()系统调用的那个进程的控制终端。守护进程，后台进程，精灵进程 |  |
| O\_TRUNC | 若文件已存在，且以读写或只写成功打开，那么先全部删除文件中的原有数据 |  |
| O\_APPEND | 以追加方式打开文件，打开文件的同时，文件指针指向文件的末尾 |  |

mode:文件的存取权限

比如：0664，0666，0755等等

最终的文件权限：mode & (~umask)

其中umask为掩码。

返回值：

成功返回新的文件描述符；出错返回-1并errno会设置错误号（指向具体出错原因）

##### close()函数

函数原型：



函数描述：

关闭一个打开的文件描述符对应的文件，同时释放这个文件描述符，并且同 时会将

缓冲区里的数据刷出去。（相当于包含了一个fflush()操作）

函数参数：

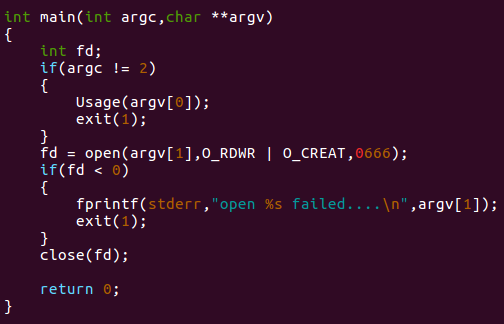
指定的文件描述符.

返回值：

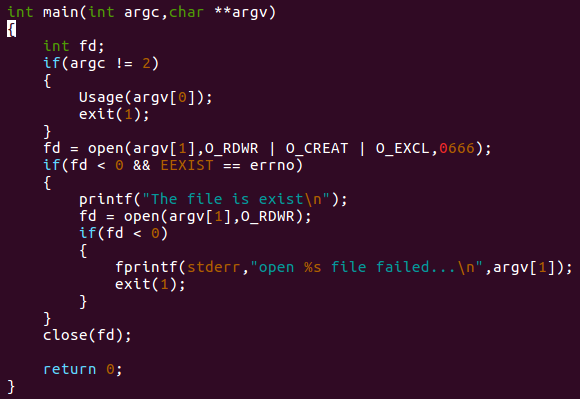
成功返回0，出错返回-1并设置一个错误号errno。

##### 例程

简便式写法：



优化式写法：



##### read()函数

函数原型：



函数描述：

read()函数从fd文件描述符指向的文件中读取count大小的数据并读取到buf缓冲区里面。

函数参数：

fd:要读取的文件的文件描述符；

buf:将数据读取到的指定的缓冲区

count:读取的数据的大小

返回值：

成功返回读取到的字节数，0表示到了文件末尾；出错返回-1并设置好相应的错误号errno。

##### write()函数

函数原型：



函数描述：

从缓冲区buf中写入count大小的数据到文件描述符fd指向的文件当中去。

参数描述：

fd:将要写入的文件的文件描述符

buf:需要写入的数据的缓冲区

count:要写入的数据的大小

返回值：

如果成功，则返回写入的字节数(0表示什么都没有写入)。如果此数字小于所请求的字节数，则不报错;例如，这可能发生在磁盘设备已被填充的情况下。

出现错误时，返回-1，并适当设置errno。

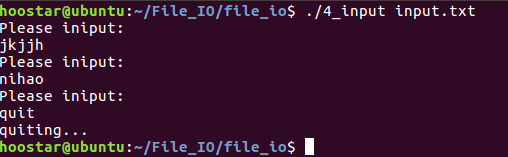
##### 例程

###### input.c

实现从键盘输入数据到指定文件中

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <strings.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #define SIZE 50  #define QUIT "quit"  void Usage(char \*s)  {  printf("Usage:\n");  printf("\t%s dst\_file\n",s);  printf("\tdst\_file:The file to write in\n");  }  int main(int argc,char \*\*argv)  {  int fd = -1;  int ret = -1;  char buf[SIZE];  if(argc != 2)  {  Usage(argv[0]);  exit(1);  }  fd = open(argv[1],O\_RDWR | O\_CREAT | O\_EXCL,0666);  if(fd < 0 && EEXIST == errno)  {  fd = open(argv[1],O\_RDWR);  if(fd < 0)  {  printf("open the dst\_file failed...\n");  exit(1);  }  }  while(1)  {  bzero(buf,SIZE);  printf("Please iniput: \n");  //小细节：这里要加换行符，不然"Please input"会一直在缓冲区里打印不出来    do{  ret = read(0,buf,SIZE-1);  }while(ret < 0 && EINTR == errno);  if(ret < 0)  {  perror("read failed\n");  continue;  }  if(!strncasecmp(QUIT,buf,strlen(QUIT)))  {  printf("quiting...\n");  break;  }    do{  ret = write(fd,buf,strlen(buf));  }while(ret < 0 && EINTR == errno);  if(ret < 0)  {  printf("write failed\n");  continue;  }  fsync(fd);  }  close(fd);  return 0;  } |

运行效果：



###### copy.c

备注：作为作业

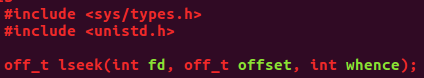
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <strings.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #undef BUFSIZ  #define BUFSIZ 20  void Usage(char \*s)  {  printf("Usage:\n");  printf("\t%s src\_file dst\_file\n",s);  printf("\tsrc\_file:The file need to be copy\n");  printf("\tdst\_file:The file need to copy on\n");  }  int main(int argc,char \*\*argv)  {  int fd\_r = -1;  int fd\_w = -1;  int ret = -1;  char buf[BUFSIZ];  if(argc != 3)  {  Usage(argv[0]);  exit(1);  }  fd\_r = open(argv[1],O\_RDONLY);  if(fd\_r < 0)  {  perror("open the src\_file failed...\n");  exit(1);  }  fd\_w = open(argv[2],O\_RDWR | O\_CREAT | O\_EXCL,0666);  if(fd\_w < 0 && EEXIST == errno)  {  fd\_w = open(argv[2],O\_RDWR);  if(fd\_w < 0)  {  perror("open the dst\_file failed...\n");  exit(1);  }  }  while(1)  {  do  {  bzero(buf,BUFSIZ);  ret = read(fd\_r,buf,BUFSIZ-1);  }while(ret < 0 && EINTR == errno);  if(ret < 0)  {  perror("read failed...\n");  exit(1);  }  do  {  ret = write(fd\_w,buf,strlen(buf));  }while(ret < 0 && EINTR == errno);  if(ret < 0)  {  perror("write failed...\n");  exit(1);  }    if(ret == 0)  {  break;  }  fsync(fd\_w);  }  close(fd\_w);  close(fd\_r);  return 0;  } |

运行效果：



##### lseek()函数

函数原型：



函数描述：

指针定位到文件某一个位置。

函数参数：

fd:文件描述符

offset:偏移量，每一读写操作所需要移动的距离，单位是字节的数量可正可负（向前移向后移）。

whence:（当前位置基点）

SEEK\_SET：当前位置为文件的开头，新位置为偏移量大小

SEEK\_CUR：当前位置为文件指针的位置，新位置为当前位置加上偏移量

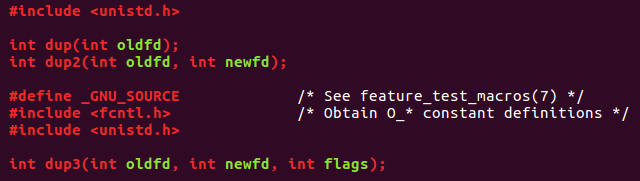
SEEK\_END：当前位置为文件结尾，新位置为文件的大小加上偏移量的大小

返回值：

成功返回文件的当前位移，出错返回-1。

##### 8）dup()函数

函数原型：



函数描述：

**dup()**：

系统调用为文件描述符oldfd创建一个副本，为新描述符使用编号最低的未使用描述符。

成功返回后，新旧文件描述符可以互换使用。它们引用相同的打开文件描述，因此共享文件偏移量和文件状态标志;例如，如果在其中一个描述符上使用lseek()修改了文件偏移量，那么另一个描述符的偏移量也会更改。

两个描述符不共享文件描述符标志(close-on-exec标志)

**dup2()**:

dup2()系统调用执行与dup()相同的任务，但它不使用编号最低的未使用文件描述符，而是使用newfd中指定的描述符编号。如果描述符newfd之前是打开的，那么它在被重用之前会被静默关闭。

注意以下几点:

\*如果oldfd不是有效的文件描述符，则调用失败，newfd不会关闭。

\*如果oldfd是一个有效的文件描述符，并且newfd的值与oldfd相同，那么dup2()不做任何事情，并返回newfd。

**dup3()**:

Dup3()与dup2()相同，不同之处在于:

调用者可以通过在flags中指定O\_CLOEXEC来强制为新的文件描述符设置“执行时关闭”标志。\*如果oldfd = newfd，则dup3()失败，错误为EINVAL。

返回值：

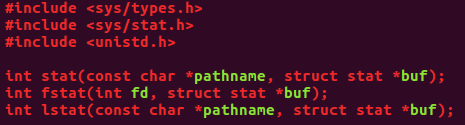
成功返回新的文件描述符，出错返回-1设置错误号errno。

例程：

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <errno.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <fcntl.h>  int main(void)  {  int fd = -1;  int newfd\_1 = -1;  int newfd\_2 = -1;  fd = open("./dup.txt",O\_RDWR | O\_CREAT | O\_EXCL,0666);  if(fd < 0 && EEXIST == errno)  {  fd = open("./dup.txt",O\_RDWR);  if(fd < 0)  {  perror("open the file failed...\n");  exit(1);  }  }  newfd\_1 = dup(fd);  printf("fd = %d\nnewfd\_1 = %d\n",fd,newfd\_1);  write(fd,"hello",strlen("hello"));  write(newfd\_1,"world",strlen("world"));  newfd\_2 = dup2(fd,7);  printf("newfd\_2 = %d\n",newfd\_2);  write(fd,"你好",strlen("你好"));  write(newfd\_2,"世界",strlen("世界"));    close(fd);  close(newfd\_1);  close(newfd\_2);  return 0;  } |

##### 9）stat()/lstat()/fstat()函数

函数原型：



函数描述：

获取文件/目录的属性



三个函数的区别：

stat()函数返回一个与此命名文件有关的信息结构

fstat()函数是获取一个已经打开了的文件的文件描述符指定的信息

lstat()函数类似于stat()函数，但若命名的文件是符号链接时，lstat()返回符号链接自身的相关信息，而不是符号链接所引用的文件的描述信息。

函数参数：

**stat()**:

pathname:文件或目录的路径；buf:用于保存函数返回的文件的有关信息。

**fstat()**:

fd:打开的文件的文件描述符；buf:用于保存函数返回的文件的有关信息。

**lstat()**:

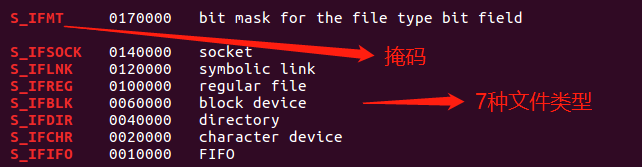
pathname:文件或目录的路径；buf:用于保存函数返回的文件的有关信息。

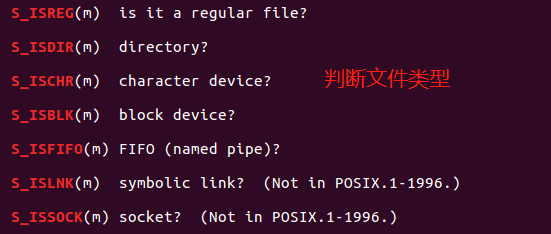
struct stat:

|  |
| --- |
| struct stat {  dev\_t st\_dev; /\* ID of device containing file \*/  ino\_t st\_ino; /\* inode number \*/  mode\_t st\_mode; /\* protection \*/  nlink\_t st\_nlink; /\* number of hard links \*/  uid\_t st\_uid; /\* user ID of owner \*/  gid\_t st\_gid; /\* group ID of owner \*/  dev\_t st\_rdev; /\* device ID (if special file) \*/  off\_t st\_size; /\* total size, in bytes \*/  blksize\_t st\_blksize; /\* blocksize for filesystem I/O \*/  blkcnt\_t st\_blocks; /\* number of 512B blocks allocated \*/  time\_t st\_atim; /\* time of last access \*/  time\_t st\_mtim; /\* time of last modification \*/  time\_t st\_ctim; /\* time of last status change \*/  }; |

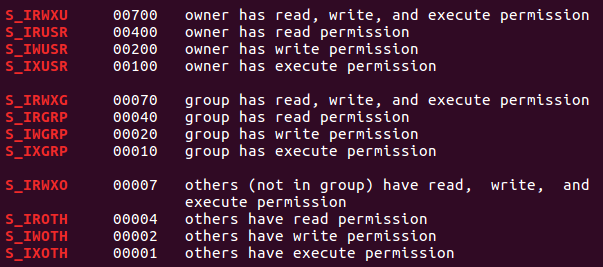
st\_mode:

1. 取得文件类型





1. 取得文件权限



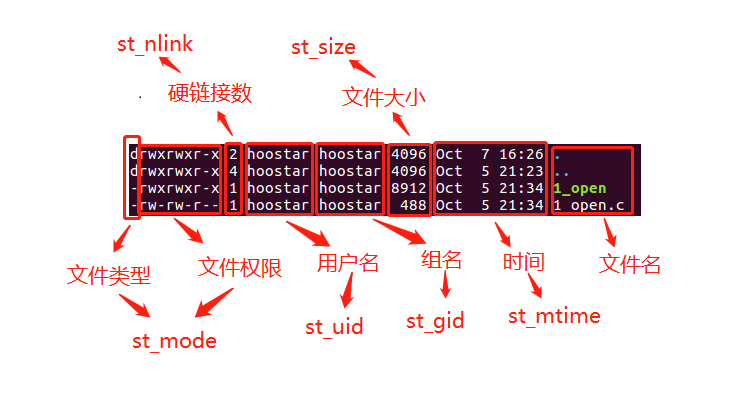
判断文件类型有两种方法：

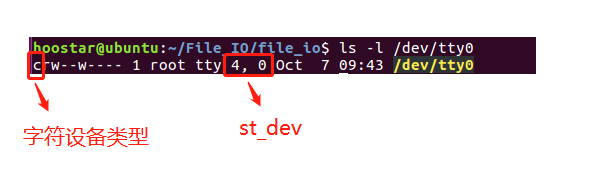
struct stat st;

if(st.st\_mode & s\_IFMT == S\_IFREG)

或者if(S\_ISREG(st.st\_mode))

文件属性分析：





获取用户名和组名：

|  |
| --- |
| **用户和密码操作函数：** /etc/passwd (只有超级用户才能读取) -------------- 通过uid获取用户名：   #include <sys/types.h>  #include <pwd.h> struct passwd \*getpwuid(uid\_t uid); struct passwd {             char   \*pw\_name;    /\* username \*/  /\* 从这里可以取到用户名\*/             char   \*pw\_passwd;   /\* user password \*/             uid\_t   pw\_uid;     /\* user ID \*/             gid\_t   pw\_gid;     /\* group ID \*/             char   \*pw\_gecos;    /\* real name \*/             char   \*pw\_dir;     /\* home directory \*/             char   \*pw\_shell;    /\* shell program \*/  }; --------------- 通过gid获取组名： SYNOPSIS        #include <sys/types.h>        #include <grp.h>         struct group \*getgrgid(gid\_t gid);  The group structure is defined in <grp.h> as follows:             struct group {              **char   \*gr\_name;  /\* group name \*/** /\* 从这里可以取到组名\*/                char   \*gr\_passwd; /\* group password \*/                gid\_t   gr\_gid;   /\* group ID \*/                char  \*\*gr\_mem;   /\* group members \*/            }; |

获取时间的操作：

|  |
| --- |
| struct tm {  int tm\_sec; /\* Seconds (0-60) \*/  int tm\_min; /\* Minutes (0-59) \*/  int tm\_hour; /\* Hours (0-23) \*/  int tm\_mday; /\* Day of the month (1-31) \*/  int tm\_mon; /\* Month (0-11) \*/  int tm\_year; /\* Year - 1900 \*/  int tm\_wday; /\* Day of the week (0-6, Sunday = 0) \*/  int tm\_yday; /\* Day in the year (0-365, 1 Jan = 0) \*/  int tm\_isdst; /\* Daylight saving time \*/  };  char tm\_buf[50];  bzero(tm\_buf,50);  time\_t m\_sec = st.st\_mtime;  struct tm \*tm = localtime(&m\_sec);  const char \*month[]={  "Jan",  "Feb",  "Mar",  "Apr",  "May",  "Jun",  "Jul",  "Aug",  "Sep",  "Oct",  "Nov",  "Dec",  };    sprintf(tm\_buf,"%s %d %d:%d",month[tm->tm\_mon],tm->tm\_mday,tm->tm\_hour,tm->tm\_min); |

对目录的操作：

|  |
| --- |
| 目录是文件的一种，是一种特殊文件   SYNOPSIS        #include <sys/types.h>        #include <dirent.h>         DIR \*opendir(const char \*name);        int closedir(DIR \*dirp);        struct dirent \*readdir(DIR \*dirp);           //多次调用会依次返回目录下面的文件或者子目录         // 所有的目录下的文件或者子目录列举完毕，返回为NULL          struct dirent {           ino\_t   d\_ino;       /\* inode number \*/           off\_t   d\_off;       /\* offset to the next dirent \*/            unsigned short d\_reclen;    /\* length of this record \*/            unsigned char  d\_type;    /\* type of file; not supported                             by all file system types \*/            char   d\_name[256]; /\* filename \*/ //目录下面的文件或子目录的名字            };   ------------编程--------------   DIR \*dp;  struct dirent \*psd;  struct stat stb;  dp = opendir (char \*path);  chdir(path);     while ((psd = readdir (dp)) != NULL) {      psd->d\_name (目录下面的文件或子目录的名字，d\_name只是一个字符串，不包含路径信息)        stat(psd->dname, &stb); } |

颜色打印

|  |
| --- |
| color.h:  ================ /\*printf color \*/ #ifndef \_\_PRINTF\_COLOR\_H\_\_ #define \_\_PRINTF\_COLOR\_H\_\_  #define NONE "\033[m" #define RED "\033[0;32;31m" #define LIGHT\_RED "\033[1;31m" #define GREEN "\033[0;32;32m" #define LIGHT\_GREEN "\033[1;32m" #define BLUE "\033[0;32;34m" #define LIGHT\_BLUE "\033[1;34m" #define DARY\_GRAY "\033[1;30m" #define CYAN "\033[0;36m" #define LIGHT\_CYAN "\033[1;36m" #define PURPLE "\033[0;35m" #define LIGHT\_PURPLE "\033[1;35m" #define BROWN "\033[0;33m" #define YELLOW "\033[1;33m" #define LIGHT\_GRAY "\033[0;37m" #define WHITE "\033[1;37m"  #define HIGHTLIGHT "\033[1m" #define UNDERLINE  "\033[4m" #define FLICKER    "\033[5m" #define INVERSE    "\033[7m"  #endif  ===============  使用：   printf( GREEN "%s 存在！\n" NONE , "test"); |

##### 例程

###### 实现简易版ls功能

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <strings.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <pwd.h>  #include <time.h>  #include <grp.h>  void Usage(char \*s)  {  printf("Usage:\n");  printf("\t%s dst\_file\n",s);  printf("\tdst\_file:the destination file need to show\n");  }  int main(int argc,char \*\*argv)  {  int i = 0;  char \*s = "xwr";  char tm\_buf[50];  char buf[BUFSIZ];  struct stat st;  char mode\_buf[10] = {0};  if(argc != 2)  {  Usage(argv[0]);  exit(1);  }    if(stat(argv[1],&st) < 0)  {  perror("error getting file properties.\n");  exit(1);  }  /\*st\_mode:获取文件的类型与权限\*/  switch(st.st\_mode & S\_IFMT)  {  case S\_IFSOCK:mode\_buf[0] = 's';break;  case S\_IFLNK:mode\_buf[0] = 'l';break;  case S\_IFREG:mode\_buf[0] = '-';break;  case S\_IFBLK:mode\_buf[0] = 'b';break;  case S\_IFDIR:mode\_buf[0] = 'd';break;  case S\_IFCHR:mode\_buf[0] = 'c';break;  case S\_IFIFO:mode\_buf[0] = 'p';break;  }  for(i = 0;i < 9;i++)  {  if(st.st\_mode & (1<<i))  {  mode\_buf[9-i] = s[i%3];  }  else  {  mode\_buf[9-i] = '-';  }  }  /\*st\_uid:获取用户名st\_gid:获取组名\*/  struct passwd \*username = getpwuid(st.st\_uid);  struct group \*grpname = getgrgid(st.st\_gid);  /\*st\_mtime:获取时间\*/  bzero(tm\_buf,50);  time\_t m\_sec = st.st\_mtime;  struct tm \*tm = localtime(&m\_sec);  const char \*month[]={  "Jan",  "Feb",  "Mar",  "Apr",  "May",  "Jun",  "Jul",  "Aug",  "Sep",  "Oct",  "Nov",  "Dec",  };    sprintf(tm\_buf,"%s %d %d:%d",month[tm->tm\_mon],tm->tm\_mday,tm->tm\_hour,tm->tm\_min);  bzero(buf,BUFSIZ);  sprintf(buf,"%s %ld %s %s %ld %s %s\n",mode\_buf,st.st\_nlink,username->pw\_name,grpname->gr\_name,st.st\_size,tm\_buf,argv[1]);  printf("%s",buf);  return 0;  } |

###### 实现改进版ls功能

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <strings.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <pwd.h>  #include <time.h>  #include <grp.h>  #include <dirent.h>  #include "color.h"  void show\_file\_info(struct stat \*st\_file,char \*name);  char \*color\_file(char \*name,struct stat \*stb);  void Usage(char \*s)  {  printf("Usage:\n");  printf("\t1、\n");  printf("\t %s\n",s);  printf("\t2、\n");  printf("\t %s dst\_file\n",s);  printf("\t dst\_file:the file need to ls\n");  }  int main(int argc,char \*\*argv)  {  int len = -1;  struct stat st;  char path[BUFSIZ];  if(argc != 1 && argc != 2)  {  Usage(argv[0]);  }  if(argc == 1)/\*当不传参数时，路径默认为当前目录下 \*/  {  strcpy(path,".");  }  else/\*当传参时就要看传的是目录还是文件\*/  {  len = strlen(argv[1]);  if(len > BUFSIZ - 1)  {  len = BUFSIZ-1;  }  strncpy(path,argv[1],len);  }  /\*获取文件或目录的属性\*/  if(stat(path,&st) < 0)  {  perror("stat(path,&st) error...\n");  exit(1);  }  /\*判断是文件还是目录\*/  if((st.st\_mode & S\_IFMT) == S\_IFDIR)  {  /\*判断是目录成功\*/  DIR \*dp;  struct dirent \*psd;  struct stat stb;  dp = opendir(path);  chdir(path);  while((psd = readdir(dp)) != NULL)  {  if(stat(psd->d\_name,&stb) < 0)  {  perror("stat(psd->d\_name,&stb)");  continue;  }    show\_file\_info(&stb,psd->d\_name);  }  closedir(dp);  }  else  {  /\*判断是文件成功\*/  show\_file\_info(&st,argv[1]);  }  return 0;  }  void show\_file\_info(struct stat \*st\_file,char \*name)  {  int i = 0;  char \*s = "xwr";  char buf[BUFSIZ];/\*注意这尽量给大一点空间\*/  char tm\_buf[50];  struct tm \*tm;  struct passwd \*uid;  struct group \*gid;  /\*st\_mode获取文件类型和权限\*/  char mode\_buf[10];  bzero(mode\_buf,10);  /\*获取文件类型\*/  switch(st\_file->st\_mode & S\_IFMT)  {  case S\_IFSOCK:mode\_buf[0] = 's';break;  case S\_IFLNK:mode\_buf[0] = 'l';break;  case S\_IFREG:mode\_buf[0] = '-';break;  case S\_IFBLK:mode\_buf[0] = 'b';break;  case S\_IFDIR:mode\_buf[0] = 'd';break;  case S\_IFCHR:mode\_buf[0] = 'c';break;  case S\_IFIFO:mode\_buf[0] = 'p';break;  }  /\*获取文件权限\*/  for(i = 0;i < 9;i++)  {  if(st\_file->st\_mode & (1<<i))  {  mode\_buf[9-i] = s[i%3];  }  else  {  mode\_buf[9-i] = '-';  }  }  /\*获取用户名和组名：st\_uid,st\_gid\*/  uid = getpwuid(st\_file->st\_uid);  gid = getgrgid(st\_file->st\_gid);  /\*获取时间st\_mtime\*/  bzero(tm\_buf,50);  time\_t cur\_sec = st\_file->st\_mtime;  tm = localtime(&cur\_sec);  const char \*month[] = {  "Jan",  "Feb",  "Mar",  "Apr",  "May",  "Jun",  "Jul",  "Aug",  "Sep",  "Oct",  "Nov",  "Dec",  };  snprintf(tm\_buf,50-1,"%s %d %d:%d",month[tm->tm\_mon],tm->tm\_mday,tm->tm\_hour,tm->tm\_min);  bzero(buf,BUFSIZ);  snprintf(buf,BUFSIZ-1,"%s %ld %s %s %ld %s %s",mode\_buf,st\_file->st\_nlink,uid->pw\_name,gid->gr\_name,st\_file->st\_size,tm\_buf,color\_file(name,st\_file));  printf("%s\n",buf);  }  char \*color\_file(char \*name,struct stat \*stb)  {  static char color\_buf[50];  bzero(color\_buf,50);  if(!name || !stb)  {  return NULL;  }  strcpy(color\_buf,name);  if(stb->st\_mode & S\_IFMT == S\_IFDIR)  {  /\*如果是目录则打印蓝色\*/  sprintf(color\_buf,LIGHT\_BLUE"%s"NONE,name);  return color\_buf;  }  if((stb->st\_mode & S\_IXUSR) || (stb->st\_mode & S\_IXGRP) || (stb->st\_mode & S\_IXOTH))  {  /\*只要文件有可执行权限就打印绿色\*/  sprintf(color\_buf,LIGHT\_GREEN"%s"NONE,name);  return color\_buf;  }  /\*普通文件什么颜色都没有\*/  return color\_buf;  } |