1.stat函数

1.1stat函数的功能

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int lstat(const char *pathname, struct stat *statbuf);
//显示软连接文件类型，用法和stat一样
int stat(const char *pathname, struct stat *statbuf);
功能：获取文件的属性信息
参数：
    @pathname:文件的路径及名字
    @statbuf:定义结构体变量，将地址传递给statbuf指针，
             代表的就是获取到的属性信息
           struct stat {
               dev_t     st_dev;         //磁盘的设备号 major(st_dev) minor(st_dev)
                                          //设备号，内核识别驱动的唯一编号
                                          // 设备号（32） = 主设备号（12） + 次设备号（20）
                                          //鼠标设备号 = 13  << 20 + 32
                                          //主设备号：那一类设备
                                          //次设备号：同类中的那个设备
               ino_t     st_ino;         //inode号，文件系统识别文件的唯一编号
               mode_t    st_mode;        /* File type and mode */
               //文件的类型  bit12-bit15
                if((st_mode & S_IFMT)==S_IFREG){
                    printf("这是普通文件\n");
                }
                   S_IFMT     0170000   获取文件类型的掩码
                   S_IFSOCK   0140000   socket
                   S_IFLNK    0120000   symbolic link
                   S_IFREG    0100000   regular file
                   S_IFBLK    0060000   block device
                   S_IFDIR    0040000   directory
                   S_IFCHR    0020000   character device
                   S_IFIFO    0010000   FIFO

              //文件的权限  bit0-bit8
                文件权限 = st_mode & 0777;

               nlink_t   st_nlink;       //硬链接数
               uid_t     st_uid;         //用户ID
               gid_t     st_gid;         //组ID
               dev_t     st_rdev;        //如果是设备文件，这里显示的就是它的设备号
                                         //如果是普通文件，这里无意义。（c字符，b块设备）
               off_t     st_size;        //文件的大小，单位是字节
               blksize_t st_blksize;     //文件系统block大小（512字节）
               blkcnt_t  st_blocks;      //文件大小，单位是block
               struct timespec st_atim;  //最后一次访问的时间
               struct timespec st_mtim;  //最后一次修改的时间
               struct timespec st_ctim;  //最后一次状态改变的时间
           };

返回值：成功返回0，失败返回-1置位错误码
```

1.2stat函数实例

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    struct stat st;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out filename\n");
        return -1;
    }

    if (stat(argv[1], &st))
        PRINT_ERR("stat error");

    switch (((st.st_mode) & S_IFMT)) {
    case S_IFSOCK:
        printf("这是套接字文件\n");
        break;
    case S_IFDIR:
        printf("这是目录文件\n");
        break;
    case S_IFREG:
        printf("这是普通文件\n");
        break;
    case S_IFBLK:
        printf("这是块设备文件\n");
        break;
    case S_IFCHR:
        printf("这是字符文件\n");
        break;
    case S_IFIFO:
        printf("这是管道文件\n");
        break;
    case S_IFLNK:
        printf("这是软连接文件\n");
        break;
    }

    printf("文件权限 = %#o\n", (st.st_mode & 0777));
    printf("硬连接数 = %ld\n", st.st_nlink);
    printf("uid = %d\n", st.st_uid);
    printf("gid = %d\n", st.st_gid);
    printf("文件大小 = %ld\n", st.st_size);

    struct tm* tm;
    if ((tm = localtime(&st.st_mtim.tv_sec)) == NULL)
        PRINT_ERR("localtime error");
    printf("%d-%02d-%02d %02d:%02d:%02d\n",
        tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday,
        tm->tm_hour, tm->tm_min, tm->tm_sec);
    return 0;
}
```

1.3stat函数的练习

练习：使用stat函数实现ls -l的功能

```c
#include <head.h>
#include <sys/sysmacros.h>
int print_file_type(mode_t mode)
{
    int flags = 0;
    switch ((mode & S_IFMT)) {
    case S_IFSOCK:
        printf("s");
        break;
    case S_IFDIR:
        printf("d");
        break;
    case S_IFREG:
        printf("-");
        break;
    case S_IFBLK:
        printf("b");
        flags = 1;
        break;
    case S_IFCHR:
        printf("c");
        flags = 1;
        break;
```

```c
    case S_IFIFO:
        printf("p");
        break;
    case S_IFLNK:
        printf("l");
        break;
    }
    return flags;
}
void print_file_mode(mode_t mode)
{
    // mode & 0777 = 0664 = 0b110 110 100   rw-rw-r--
    mode_t m = mode & 0777; // 0664 & 0400
    for (int i = 0; i < 9; i++) {
        if ((m << i) & 0400) {
            switch (i) {
            case 0:
            case 3:
            case 6:
                putchar('r');
                break;
            case 1:
            case 4:
            case 7:
                putchar('w');
                break;
            case 2:
            case 5:
            case 8:
                putchar('x');
                break;
            }

        } else {
            putchar('-');
        }
    }
}
int print_file_time(time_t ts)
{
    struct tm* tm;
    if ((tm = localtime(&ts)) == NULL)
        PRINT_ERR("localtime error");
    printf("%d月  %d %02d:%02d",
        tm->tm_mon + 1, tm->tm_mday,
        tm->tm_hour, tm->tm_min);
    return 0;
}
void print_file_info(struct stat* st, const char* name)
{
    int flags;
    // 1.打印文件类型
    flags = print_file_type(st->st_mode);
    // 2.打印文件权限
    print_file_mode(st->st_mode);
    // 3.打印硬链接数
    printf(" %ld ", st->st_nlink);
    // 4.打印用户和组id
    printf("%s %s ", getpwuid(st->st_uid)->pw_name, getgrgid(st->st_gid)->gr_name);
    // 5.文件的大小
    if(flags){
        //设备文件
        printf("%d, %d ",major(st->st_rdev),minor(st->st_rdev));
    }else{
        //普通文件
        printf("%ld ", st->st_size);
    }
    // 6.文件的修改时间
    print_file_time(st->st_mtim.tv_sec);
    // 7.文件名
    printf(" %s\n", name);
}

int main(int argc, const char* argv[])
{
    struct stat st;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out filename\n");
        return -1;
    }

    if (lstat(argv[1], &st))
        PRINT_ERR("stat error");
```

```
    print_file_info(&st, argv[1]);

    return 0;
}
```

2.getpwuid/getgrgid函数

2.1getpwuid函数功能

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
功能：根据uid获取用户信息结构体
参数：
    @uid:用户的id
返回值：成功返回passwd结构体指针，失败返回NULL,置位错误码
   struct passwd {
        char   *pw_name;       //用户名
        char   *pw_passwd;     //密码占位符
        uid_t   pw_uid;        //uid
        gid_t   pw_gid;        //gid
        char   *pw_gecos;      //用户信息
        char   *pw_dir;        //用户主目录
        char   *pw_shell;      //命令行解析器
   };
```

2.2getpwuid函数实例

```
#include <head.h>

int main(int argc,const char * argv[])
{
    int uid = 1000;
    struct passwd *pwd;

    if((pwd = getpwuid(uid))==NULL)
        PRINT_ERR("getpwduid error");

    printf("用户名 = %s\n",pwd->pw_name);
    printf("密码 = %s\n",pwd->pw_passwd);
    printf("uid = %d\n",pwd->pw_uid);
    printf("gid = %d\n",pwd->pw_gid);
    printf("登录名 = %s\n",pwd->pw_gecos);
    printf("用户主目录 = %s\n",pwd->pw_dir);
    printf("命令行解析器 = %s\n",pwd->pw_shell);

    return 0;
}
```

2.3getgrgid函数功能

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

struct group *getgrgid(gid_t gid);
功能：根据gid获取组相关信息
参数：
    @gid:组id
返回值：成功返回group结构体指针，
        失败返回NULL,置位错误码
    struct group {
        char   *gr_name;       //组名
        char   *gr_passwd;     //组密码占位符
        gid_t   gr_gid;        //组ID
        char  **gr_mem;        //组内的成员
    };
```

2.4getgrgid函数实例

```
#include <head.h>

int main(int argc,const char * argv[])
{
    int gid = 4;
    struct group *grp;

    if((grp = getgrgid(gid))==NULL)
```

```
        PRINT_ERR("getgrgid error");

    printf("组名 = %s\n",grp->gr_name);
    printf("密码 = %s\n",grp->gr_passwd);
    printf("gid = %d\n",grp->gr_gid);

    for(int i=0;grp->gr_mem[i]!=NULL;i++){
        printf("%s ",grp->gr_mem[i]);
    }
    printf("\n");

    return 0;
}
```

3.目录操作

3.1opendir函数

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
功能：打开目录
参数：
    @name:目录名
返回值：成功返回DIR*结构体指针，
        失败返回NULL，置位错误码
```

3.2readdir函数

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
功能：从打开的目录中读文件
参数：
    @dirp:打开目录的指针
返回值：成功返回dirent结构体指针，
        如果读取到目录的结尾返回NULL，错误码不会被置位
        失败返回NULL，置位错误码
    struct dirent {
       ino_t          d_ino;       //读取到文件的inode号
       unsigned short d_reclen;    //当前结构体大小
       unsigned char  d_type;      //文件的类型
              DT_BLK      This is a block device.
              DT_CHR      This is a character device.
              DT_DIR      This is a directory.
              DT_FIFO     This is a named pipe (FIFO).
              DT_LNK      This is a symbolic link.
              DT_REG      This is a regular file.
              DT_SOCK     This is a UNIX domain socket.
              DT_UNKNOWN  The file type could not be determined.
       char   d_name[256]; //文件名字
    };
```

3.3closedir函数

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
功能：关闭目录
参数：
    @dirp:目录指针
返回值：成功返回0，失败返回-1置位错误码
```

3.4目录操作实例

```
#include <head.h>

int main(int argc, const char* argv[])
{
    DIR* dir;
    struct dirent* dt;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out dir\n");
        return -1;
    }

    if ((dir = opendir(argv[1])) == NULL)
        PRINT_ERR("opendir error");

    while ((dt = readdir(dir)) != NULL) {
```

```
        printf("%s ", dt->d_name);
        switch (dt->d_type) {
        case DT_BLK:
            printf("这是块设备文件 ");
            break;
        case DT_CHR:
            printf("这是字符设备文件 ");
            break;
        case DT_DIR:
            printf("这是目录文件 ");
            break;
        case DT_FIFO:
            printf("这是管道文件 ");
            break;
        case DT_LNK:
            printf("这是软连接文件 ");
            break;
        case DT_REG:
            printf("这是普通文件 ");
            break;
        case DT_SOCK:
            printf("这是套接字文件 ");
            break;
        }

        printf("%ld ",dt->d_ino);
        printf("%d\n",dt->d_reclen);
    }

    closedir(dir);
    return 0;
}
```

3.5目录操作练习

向一个程序中输入文件名，判断指定目录下是否有这个文件，

如果有这个文件，将这个文件的属性信息输出。如果不存在

输出不存在即可。

./a.out filename

> 注：例如指定的目录是/home/linux/work/

```
#include <head.h>
#include <sys/sysmacros.h>
#define SERVER_PATH "/home/linux/work/day3"
int print_file_type(mode_t mode)
{
    int flags = 0;
    switch ((mode & S_IFMT)) {
    case S_IFSOCK:
        printf("s");
        break;
    case S_IFDIR:
        printf("d");
        break;
    case S_IFREG:
        printf("-");
        break;
    case S_IFBLK:
        printf("b");
        flags = 1;
        break;
    case S_IFCHR:
        printf("c");
        flags = 1;
        break;
    case S_IFIFO:
        printf("p");
        break;
    case S_IFLNK:
        printf("l");
        break;
    }
    return flags;
}
void print_file_mode(mode_t mode)
{
    // mode & 0777 = 0664 = 0b110 110 100   rw-rw-r--
    mode_t m = mode & 0777; // 0664 & 0400
    for (int i = 0; i < 9; i++) {
        if ((m << i) & 0400) {
            switch (i) {
```

```
            case 0:
            case 3:
            case 6:
                putchar('r');
                break;
            case 1:
            case 4:
            case 7:
                putchar('w');
                break;
            case 2:
            case 5:
            case 8:
                putchar('x');
                break;
            }

        } else {
            putchar('-');
        }
    }
}
int print_file_time(time_t ts)
{
    struct tm* tm;
    if ((tm = localtime(&ts)) == NULL)
        PRINT_ERR("localtime error");
    printf("%d月  %d %02d:%02d",
        tm->tm_mon + 1, tm->tm_mday,
        tm->tm_hour, tm->tm_min);
    return 0;
}
void print_file_info(struct stat* st, const char* name)
{
    int flags;
    // 1.打印文件类型
    flags = print_file_type(st->st_mode);
    // 2.打印文件权限
    print_file_mode(st->st_mode);
    // 3.打印硬链接数
    printf(" %ld ", st->st_nlink);
    // 4.打印用户和组id
    printf("%s %s ", getpwuid(st->st_uid)->pw_name, getgrgid(st->st_gid)->gr_name);
    // 5.文件的大小
    if(flags){
        //设备文件
        printf("%d, %d ",major(st->st_rdev),minor(st->st_rdev));
    }else{
        //普通文件
        printf("%ld ", st->st_size);
    }
    // 6.文件的修改时间
    print_file_time(st->st_mtim.tv_sec);
    // 7.文件名
    printf(" %s\n", name);
}

int main(int argc, const char* argv[])
{
    DIR* dir;
    struct dirent* dt;
    struct stat st;
    char file[256] = {0};
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out filename\n");
        return -1;
    }

    if ((dir = opendir(SERVER_PATH)) == NULL)
        PRINT_ERR("opendir error");

    while ((dt = readdir(dir)) != NULL) {
        if (strcmp(argv[1], dt->d_name) == 0) {
            snprintf(file,sizeof(file),"%s/%s",SERVER_PATH,argv[1]);

            if(lstat(file,&st))
                PRINT_ERR("get stat error");

            print_file_info(&st, argv[1]);
            closedir(dir);
            return 0;
        }
    }
```

```
125     printf("查询的文件不存在，请重试\n");
126     closedir(dir);
127     return -1;
128 }
```

4.库的制作及使用

4.1库的简介

库文件其实是一个二进制文件，库文件是有xxx.c（这里没有main函数）编译而来的。

linux系统中的库分为动态库和静态库。库文件可以被用户使用，但是用户看不到库中

的函数的实现。

Linux系统:

动态库：libxxx.so　　# (lib前缀，.so是后缀，xxx库名)

静态库：libxxx.a　　# (lib前缀，.a是后缀，xxx库名)

4.2静态库

4.2.1静态库的特点

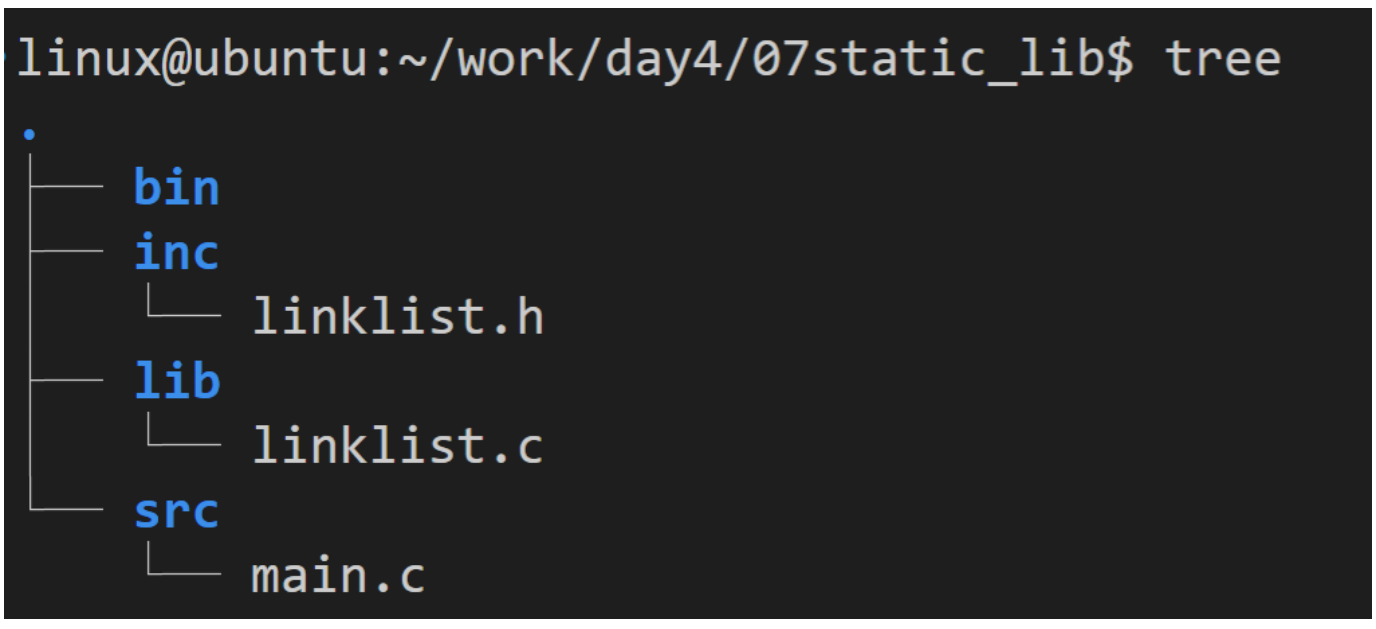
静态库：静态库的格式是libxxx.a的格式，它是由xxx.c文件编译而来的。用户在使用

静态库的时候，会将库中函数的实现和用户代码的实现最终编译生成一个可执行程序。

可执行程序的体积比较大，但是在运行可执行程序的时候不需要这个库文件了。所以

静态库的程序运行的效率比较高。更新的时候比较麻烦。

4.2.2静态库的制作



`gcc -c linklist.c -o linklist.o -I ../inc`

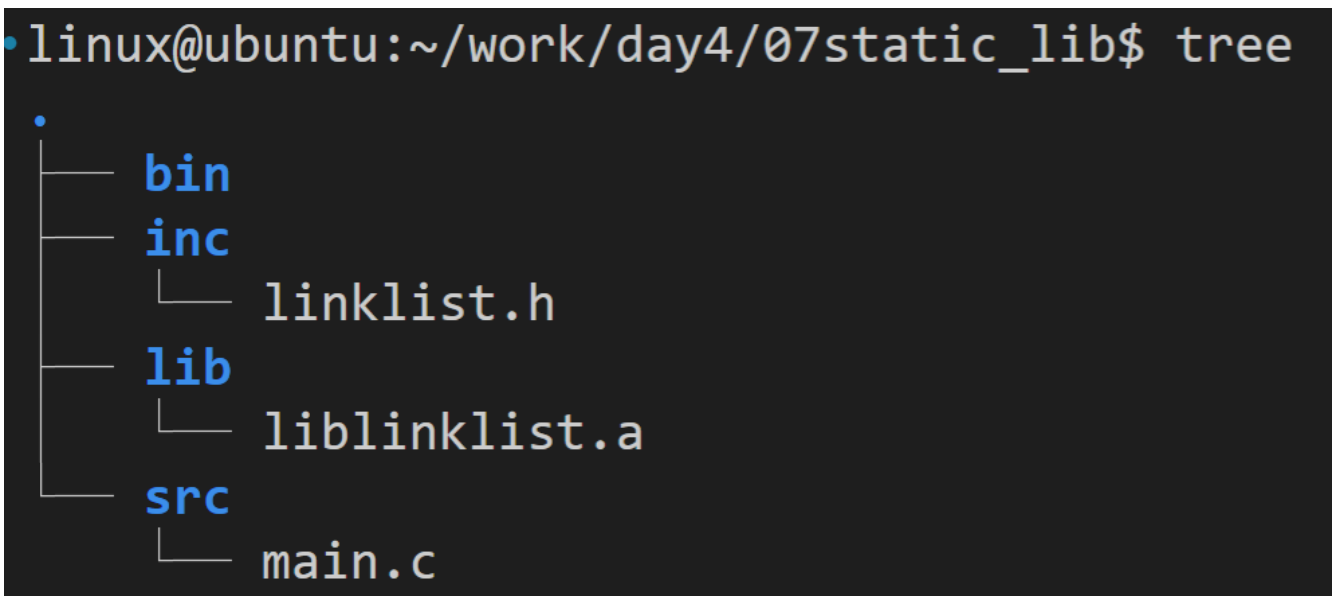
//将linklist.c只编译不链接生成.o文件

//-I ../inc 指定头文件路径

`ar -cr liblinklist.a linklist.o`

// ar制作静态库的命令

// -c 创建库文件 -r将函数放到库文件中



4.2.3静态库的使用

```
1 -L 指定库的路径
2 -l 链接库
3 -I 指定头文件路径
```

```
gcc main.c -L ../lib -llinklist -I ../inc -o ../bin/a.out
```

> 注：
>
> -L ../lib 表示使用的库在../lib目录下
>
> -llinklist 表示使用../lib/liblinklist.a的库文件
>
> -I ../inc 指定linklist.h头文件的路径
>
> -o ../bin/a.out 将编译后的可执行程序放在../bin目录下，并取名为a.out

./a.out就能看到现象

4.3动态库

4.3.1动态库的特点

动态库：动态库的格式是libxxx.so的格式，它是由xxx.c文件编译而来的。用户在使用动态库的时候，会将库中函数的符号表和用户代码的实现最终编译生成一个可执行程序。可执行程序的体积比较小，但是在运行可执行程序的时候需要在libxxx.so库中找函数的实现。依赖动态库的程序运行的效率比较低。更新的时候比较容易。动态库文件可以同时被多个程序同时使用，所以动态库又叫共享库。

4.3.2动态库的制作

```
linux@ubuntu:~/work/day4/08dynamic_lib$ tree
.
├── bin
├── inc
│   └── linklist.h
├── lib
│   └── linklist.c
└── src
    └── main.c
```

```
gcc -fPIC -shared linklist.c -o liblinklist.so  -I ../inc
```

// -fPIC :生成与位置无关的库文件

// -shared:生成共享库的标识

//上述命令生成的动态库liblinklist.so

```
linux@ubuntu:~/work/day4/08dynamic_lib$ tree
.
├── bin
├── inc
│   └── linklist.h
├── lib
│   └── liblinklist.so
└── src
    └── main.c
```

4.3.3动态库的使用

```
-L 指定库的路径
-l 链接库
-I 指定头文件路径
```

```
gcc main.c -L ../lib -llinklist -I ../inc -o ../bin/a.out
```

4.3.4指定动态库路径

在编译完之后，运行的时候出现如下错误：

```
linux@ubuntu:~/work/day4/08dynamic_lib/bin$ ./a.out
./a.out: error while loading shared libraries: liblinklist.so: cannot open shared object file: No such file or directory
```

错误原因：在运行的时候找不到动态库。解决办法有如下三种。

1. 将liblinklist.so移动到系统库路径下

   sudo mv liblinklist.so /lib

```
linux@ubuntu:~/work/day4/08dynamic_lib/bin$ ./a.out
-10-5-20-40-
-5-10-20-40-
```

2. 临时指定系统库路径（当前终端）

   export LD_LIBRARY_PATH=../lib

```
linux@ubuntu:~/work/day4/08dynamic_lib/bin$ ./a.out
-10-5-20-40-
-5-10-20-40-
```

3. 通过系统配置文件指定库路径

   sudo vi /etc/ld.so.conf.d/libc.conf

```
# libc default configuration
/usr/local/lib
/home/linux/work/day4/08dynamic_lib/lib/
```

sudo ldconfig 上配置文件生效

```
linux@ubuntu:~/work/day4/08dynamic_lib/bin$ ./a.out
-10-5-20-40-
-5-10-20-40-
```