

# 实验6、系统命令的实现

- 实验目的

- ① 掌握Linux目录操作方法，包括打开目录、关闭目录、读取目录文件
- ② 掌握Linux文件属性获取方法，包括三个获取Linux文件属性的函数、文件属性解析相关的宏
- ③ 掌握POSIX与ANSI C文件I/O操作方法，包括打开文件、关闭文件、创建文件、读写文件、定位文件

- 实验内容

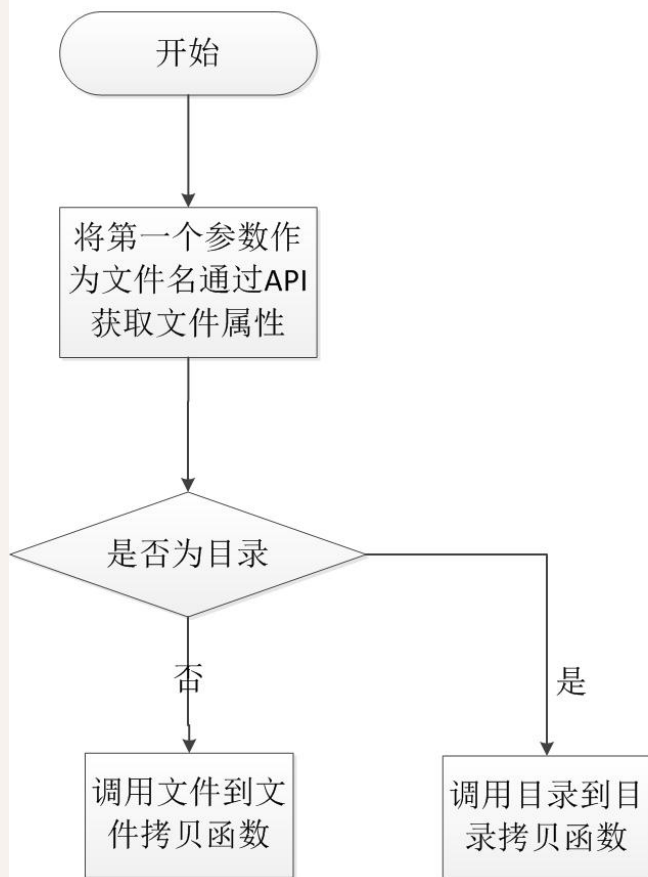
利用POSIX API（文件操作也可以使用ANSI C标准I/O库）编程实现`cp -r`命令，支持将源路径（目录）中的所有文件和子目录，以及子目录中的所有内容，全部拷贝到目标路径（目录）中。

## 实验内容：cp命令与命令行参数

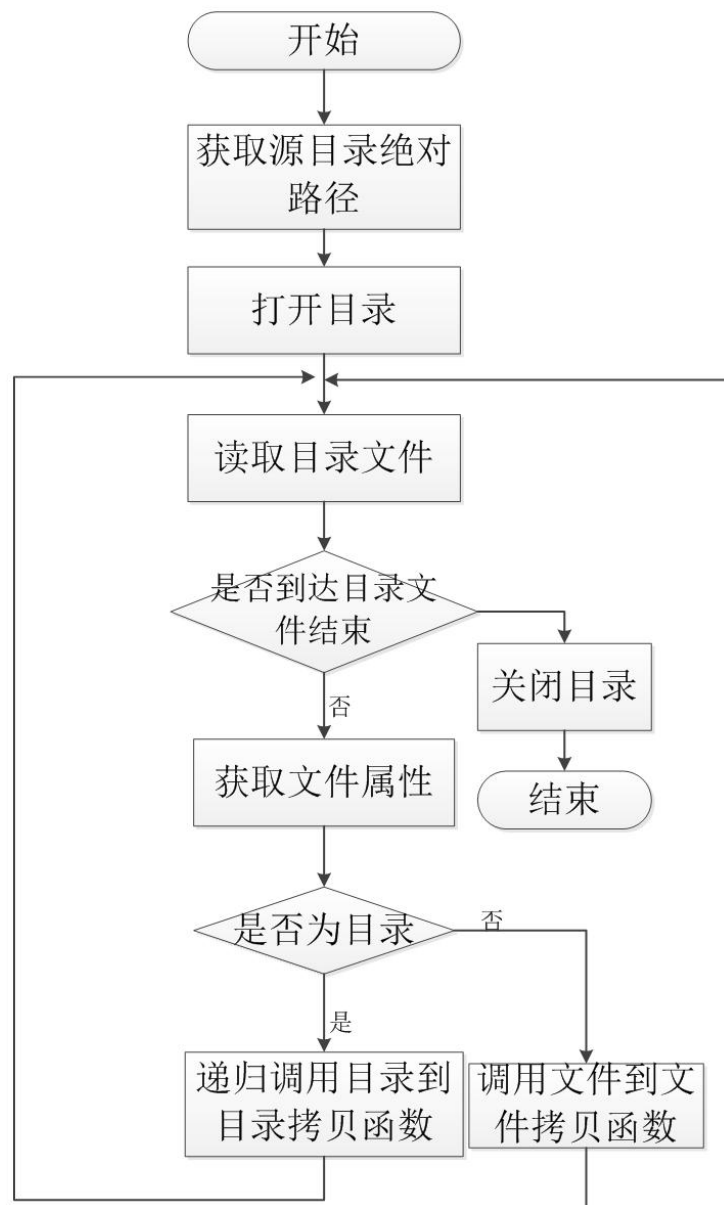
- UNIX/Linux中在shell中输入命令名（可执行文件名）来启动程序，在命令名（可执行文件名）之后可以跟随一系列字符串（通过空格分割），这些字符串就是命令行参数
- **cp** [参数] <源文件路径> <目标文件路径>  
cp /usr/local/src/main.c /root/main.c （文件到文件复制）  
cp /usr/local/src/main.c /root （文件到目录复制）  
cp -r /usr/local/src /root （递归复制，用于目录到目录的复制）

# 总体程序流程

main函数流程图



目录到目录拷贝流程图



## 任务分解1：文件到文件拷贝

- 利用POSIX API在Linux系统上编写应用程序，仿写cp命令的部分功能，将源文件复制到另外一个文件或复制到另外一个目录。源文件路径和目标文件路径通过命令行参数来指定

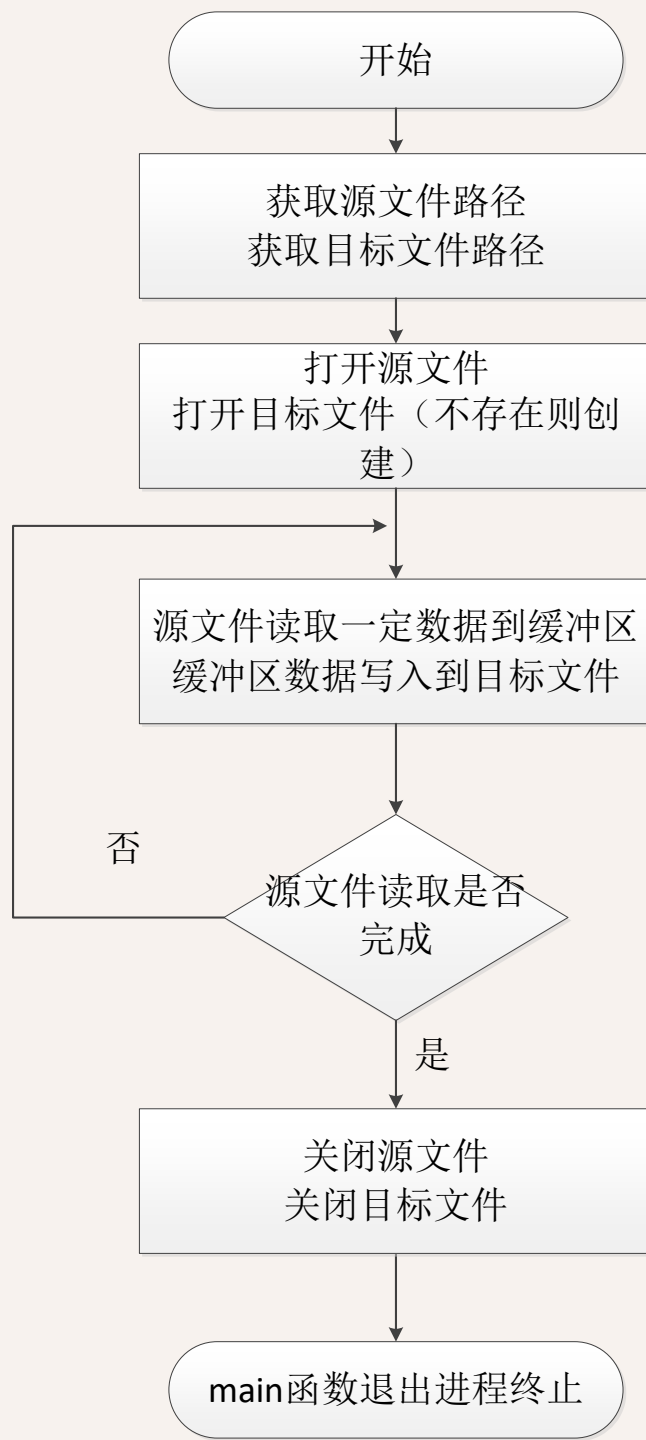
1、将test1.txt复制成test2.txt：

```
[test@linux test]$ ./mycp /home/test1.txt  
/usr/test2.txt
```

2、将test1.txt复制到/tmp目录中：

```
[test@linux test]$ ./mycp /home/test1.txt /tmp(目  
录)
```

# 程序流程



# 实验原理：应用程序命令行参数获取

- UNIX/Linux中C语言应用程序的启动函数是main
- 操作系统通过C启动例程来启动C程序，启动例程会从标准输入获取应用程序的命令行参数，并且将这些参数传递给main函数

# 实验原理：应用程序命令行参数获取

- main函数定义

```
int main(int argc, char* argv[])
```

- 形式参数：

argc：整形，命令行输入参数的个数

argv：字符串数组，以字符串形式存储的  
命令行参数



# 演示程序

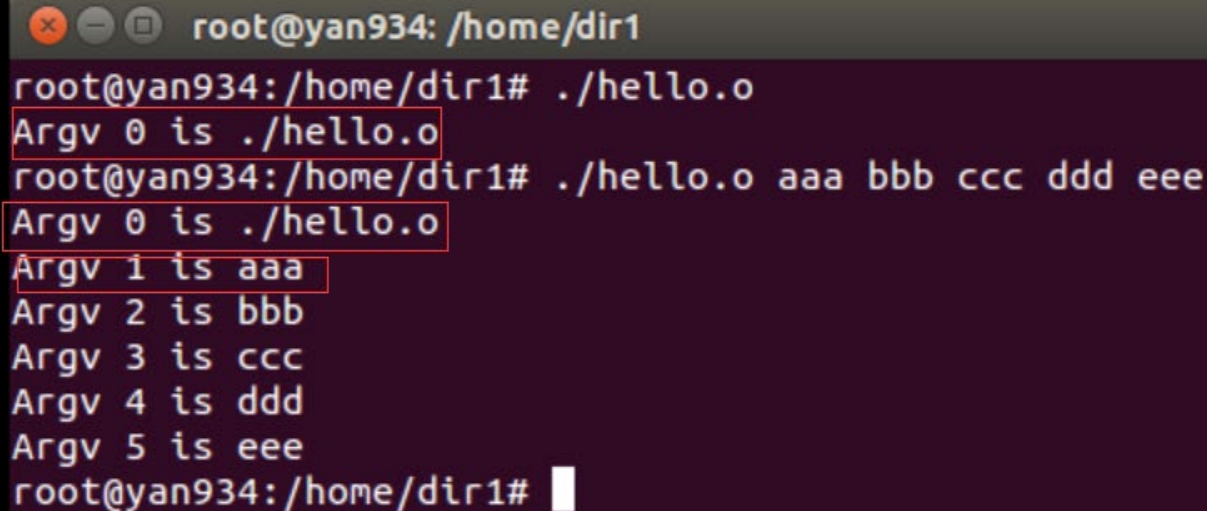
```
#include <stdio.h>

int main(int argc,char *argv[])
{
    int i;
    for(i=0;i<argc;i++)
        printf("Argv %d is %s.\n",i,argv[i]);
    return 0;
}
```

加入将上述代码编译为**hello.o**，在命令行中分别按照两种情况执行：

1、 ./hello.o

# 演示效果



```
root@yan934: /home/dir1
root@yan934:/home/dir1# ./hello.o
Argv 0 is ./hello.o
root@yan934:/home/dir1# ./hello.o aaa bbb ccc ddd eee
Argv 0 is ./hello.o
Argv 1 is aaa
Argv 2 is bbb
Argv 3 is ccc
Argv 4 is ddd
Argv 5 is eee
root@yan934:/home/dir1#
```

The image shows a terminal window with a dark background. The title bar at the top reads 'root@yan934: /home/dir1'. The terminal content shows two commands being executed. The first command is './hello.o', which outputs 'Argv 0 is ./hello.o'. The second command is './hello.o aaa bbb ccc ddd eee', which outputs 'Argv 0 is ./hello.o', 'Argv 1 is aaa', 'Argv 2 is bbb', 'Argv 3 is ccc', 'Argv 4 is ddd', and 'Argv 5 is eee'. Each output line is highlighted with a red rectangular box. The prompt 'root@yan934:/home/dir1#' is visible at the end of the last line.

**Argument 0**是可执行文件名本身

**Argument 1**开始才是真正的命令行参数，以字符串的形式传递（空格作为命令行参数之间的分隔）

# 实验原理：打开文件

头文件：fcntl.h

```
int open( const char *pathname, int oflag, ...);
```

- 该函数打开或创建一个文件。其中第二个参数oflag说明打开文件的选项（第三个参数是变参，仅当创建新文件时才使用）

**O\_RDONLY:** 只读打开;

**O\_WRONLY:** 只写打开;

**O\_RDWR:** 读、写打开;

**O\_APPEND:** 每次写都加到文件尾;

**O\_CREAT:** 若此文件不存在则创建它，此时需要第三个参数mode，该参数约定了所创建文件的权限，计算方法为 $\text{mode} \& \sim \text{umask}$

**O\_EXCL:** 如同时指定了O\_CREAT，此指令会检查文件是否存在，若不存在则建立此文件；若文件存在，此时将出错。

**O\_TRUNC:** 如果此文件存在，并以读写或只写打开，则文件长度0

- 返回值是打开文件的文件描述符

# 实验原理：读文件

头文件unistd.h

```
ssize_t read( int filedes, void *buf, size_t nbytes);
```

- read函数从打开的文件中读数据  
如读取成功，返回实际读到的字节数。一般情况下实际读出的字节数等于要求读取的字节数，但也有例外：读普通文件时，在读到要求字节数之前就到达文件尾  
如已到达文件的末尾或无数据可读，返回0（**可以作为文件读取是否完成的判断条件！**）  
如果出错，返回-1
- 读操作完成后，文件的当前位置将从读之前的位置加上实际读的字节数。

# 实验原理：写文件

头文件unistd.h

```
ssize_t write( int filedes, const void *buf, size_t nbytes);
```

- **write**函数向打开的文件中写数据  
写入成功返回实际写入的字节数，通常与要求写入字节数相同  
写入出错返回-1，出错的原因可能是磁盘满、没有访问权限、或写超过文件长度限制等等
- 写操作完成后，文件的当前位置将从写之前的位置加上实际写的字节数。

# 实验原理：关闭文件

头文件unistd.h

```
int close( int fildes );
```

- 该函数关闭打开的一个文件
- 关闭文件后，就不能通过该文件描述符操作该文件了

# 实验原理：文件定位

头文件unistd.h

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

- 进程中每打开一个文件都有一个与其相关联的“文件当前位置”
- 打开文件时，文件当前位置默认为文件头（0），如果指定了O\_APPEND选项则文件当前位置变为文件尾（文件长度），
- lseek函数用于设置或查询文件当前位置

# 实验原理：文件定位

- 对参数的解释与参数whence的值有关：

若whence是SEEK\_SET,则将该文件当前位置设置为文件头+offset（以字节为单位）

若whence是SEEK\_CUR,则将该文件当前位置设置为文件当前位置+offset（以字节为单位）

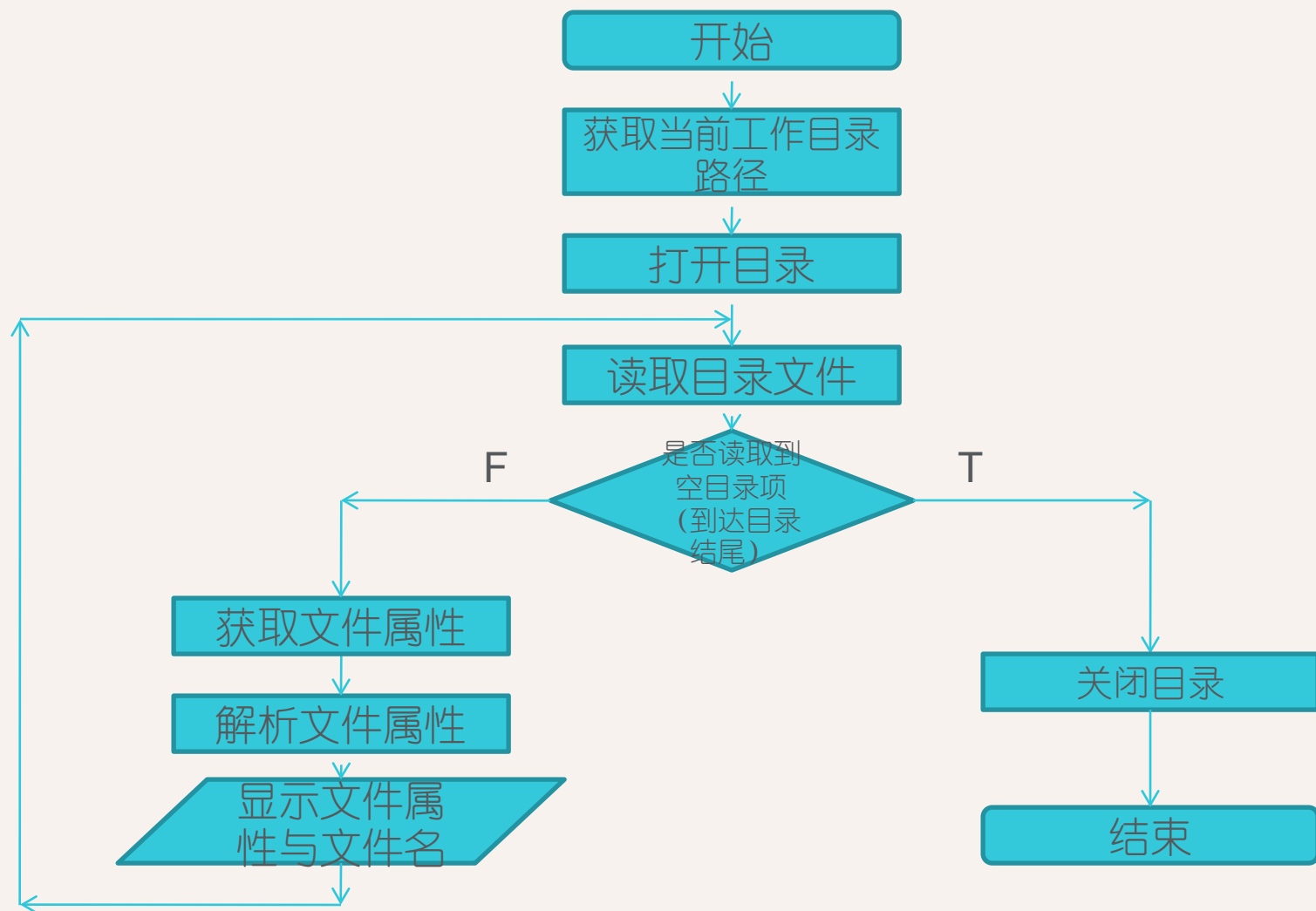
若whence是SEEK\_END,则将该文件当前位置设置为文件尾+offset个字节（以字节为单位）

- offset可正可负

```
//come from /usr/include/unistd.h↵
/* Values for the WHENCE argument to lseek.  */↵
I
#ifdef _STDIO_H      /* <stdio.h> has the same definitions. */↵
# define SEEK_SET    0  /* Seek from beginning of file.  */    //文件起始位置↵
# define SEEK_CUR    1  /* Seek from current position.  */    //当前位置↵
# define SEEK_END    2  /* Seek from end of file.  */    //文件结束位置↵
```



## 任务分解2：目录的遍历及文件属性获取



# 实验原理-获取当前工作路径

- 常用函数：getcwd, get\_current\_dir\_name
- 头文件：unistd.h
- 函数定义：

**char \*getcwd(char \*buf, size\_t size)**

- 将当前的工作目录绝对路径字符串复制到参数buf 所指的缓冲区，参数size 为缓冲区大小
- 若参数buf 为NULL，参数size 为0，则函数根据路径字符串的长度自动分配缓冲区，并将分配的路径字符串缓冲区指针作为函数返回值（该内存区需要手动释放）
- 失败返回NULL

**char \*get\_current\_dir\_name(void)**

- 成功返回路径字符串缓冲区指针（该内存区需要手动释放），失败返回NULL

# 实验原理-打开关闭目录

- 常用函数： `opendir`, `closedir`
- 头文件： `dirent.h`
- 函数定义：

**`DIR * opendir(const char * name);`**

- 打开参数 `name` 指定的目录，并使一个目录流与它关联
- 目录流类似于C库函数中的文件流
- 失败返回 `NULL`

**`int closedir(DIR *dir);`**

- 关闭指定目录流，释放相关数据结构
- 成功返回 `0`；失败返回 `-1`

# 实验原理-读取目录文件

- 常用函数：readdir
- 头文件：sys/types.h；dirent.h
- 函数定义：

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

- 读取目录流标识的目录文件
- 目录文件是一系列目录项的列表，每执行一次readdir，该函数返回指向当前读取目录项结构的指针
- 如果到达目录结尾或者有错误发生则返回NULL

- 范例代码（目录的遍历）

```
1  if((currentdir = opendir(buf)) == NULL)
2  {
3      printf("open directory fail\n");
4      return 0;
5  }
6  else
7  {
8      printf("file in directory include:\n")
9      while((currentdp = readdir(currentdir)) != NULL)
10         printf("%s ", currentdp->d_name);
11  }
```

- A
- B
- C

# 实验原理-读取目录文件

## ● 重要数据结构

struct dirent

```
{  
    ino_t d_ino; i节点号  
    off_t d_off; 在目录文件中的偏移  
    unsigned short d_reclen; 文件名长度  
    unsigned char d_type; 文件类型  
    char d_name[256]; 文件名  
};
```



```
root@xrxy-virtual-machine:~# ls -al  
total 160  
drwx----- 23 root root 4096 2014-01-02 06:19 .  
drwxr-xr-x 23 root root 4096 2012-02-23 09:56 ..  
-rwxr-xr-x 1 root root 7255 2013-12-31 22:43 a.out  
-rw----- 1 root root 1963 2014-01-01 07:31 .bash_history  
-rw-r--r-- 1 root root 3106 2011-07-08 13:13 .bashrc  
drwx----- 9 root root 4096 2013-12-31 22:39 .cache  
drwx----- 9 root root 4096 2014-01-01 02:41 .config  
drwx----- 3 root root 4096 2012-02-23 10:08 .dbus  
drwxr-xr-x 2 root root 4096 2012-02-23 10:08 Desktop
```

# 实验原理-获取文件属性

- 常用函数： `stat`， `lstat`

- 头文件： `sys/stat.h`

- 函数定义：

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

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

- 两个函数参数相同，功能类似
- 读取`path`参数所指定文件的文件属性并将其填充到`buf`参数所指向的结构体中
- 对于符号链接文件，`lstat`返回符号链接文件本身的属性，`stat`返回符号链接引用文件的文件属性

# 实验原理-文件属性结构体定义

- 重要数据结构

```
struct stat {
```

```
    mode_t    st_mode;    文件类型与访问权限
```



```
    ino_t     st_ino;     i节点号
```

```
    dev_t     st_dev;     文件使用的设备号
```

```
    dev_t     st_rdev;     设备文件的设备号
```

```
    nlink_t   st_nlink;   文件的硬链接数
```



```
    uid_t     st_uid;     文件所有者用户ID
```



```
    gid_t     st_gid;     文件所有者组ID
```



```
    off_t     st_size;     文件大小（以字节为单位）
```

```
    time_t    st_atime;     最后一次访问该文件的时间
```



```
    time_t    st_mtime;     最后一次修改该文件的时间
```



```
    time_t    st_ctime;     最后一次改变该文件状态的时间
```

```
    blksize_t st_blksize;   包含该文件的磁盘块的大小
```

```
    blkcnt_t  st_blocks;    该文件所占的磁盘块 数
```

```
};
```

# 实验原理-文件类型与权限位定义

- 重要数据结构

`mode_t st_mode;`

- 无符号整数，其低16位定义如下

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					U	G	T	R	W	X	R	W	X	R	W	X
文件类型域					文件特殊属性域			所有者权限域			组权限域			其他用户权限域		



# 实验原理-判定文件类型宏

- 是否为普通文件: `S_ISREG(st_mode)`  
`#define S_IFMT 0170000`  
`#define S_IFREG 0100000`  
`#define S_ISREG(m) (((m) & S_IFMT) == S_IFREG)`
- 是否为目录文件 `S_ISDIR(st_mode)`
- 是否为字符设备 `S_ISCHR(st_mode)`
- 是否为块设备 `S_ISBLK(st_mode)`
- 是否为FIFO `S_ISFIFO(st_mode)`
- 是否为套接字 `S_ISSOCK(st_mode)`
- 是否为符号连接 `S_ISLNK(st_mode)`

# 代码示例

```
int main(int argc, char *argv[])
{
    int            i;
    struct stat    buf;
    char           *ptr;
    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            err_ret("lstat error");
            continue;
        }
    }
}
```

# 代码示例

```
if(S_ISREG(buf.st_mode)) ptr = "regular";
else if (S_ISDIR(buf.st_mode)) ptr = "directory";
else if (S_ISCHR(buf.st_mode)) ptr = "character special";
else if (S_ISBLK(buf.st_mode)) ptr = "block special";
else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
else if (S_ISLNK(buf.st_mode)) ptr = "symbolic link";
else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
else ptr = "** unknown mode **";
printf("%s\n", ptr);
}
exit(0);
```