文件属性获取

int stat(const char *pathname, struct stat *buf)

int fstat(int fd, struct stat *buf)

int lstat(const char * pathname, struct stat *buf)

int fstatat(int fd, const char *pathname, struct stat *buf, int flag)

成功返回0，出错返回-1

1. stat和fstat和flag没有设置为AT_STMLINK_NOFOLLOW的fstat都穿透符号链接

2.lstat不穿透符号链接

3.对于fstatat，如果pathname是相对目录，那么就会计算相对于fd的实际目录，如果pathname是绝对目录，那么fd将会被忽略。特别的，fd设置为AT_FDCWD, 代表参考位置是该进程的工作目录。

文件属性结构

```
struct stat
{
    mode_t st_mode;
    //文件类型和拥有者 、同组者 、其他人的访问权限，设置用户ID和组ID位
    ino_t st_ino;
    //i节点编号
    dev_t st_dev;
    //设备号1
    dev_t st_rdev;
    //特殊设备号
    nlink_t st_nlink;
    //连接数目
    uid_t st_uid;
    //文件拥有者的用户ID
    gid_t st_gid;
    //文件所属组的用户ID
    off_t st_size;
    //文件的大小
    struct timespec st_atime;
    //最后访问时间
    struct timespec st_mtime;
    //最后修改时间
    struct timespec st_ctime;
    //最后状态改变时间
    blksize_t st_blksize;
    //最佳的磁盘块大小
    blkcnt_t st_blocks;
    //磁盘块的数目
}
```

文件属性_文件类型

1. 文件都有哪些类型？

普通文件 、目录 、块设备文件 、字符设备文件 、符号链接 、FIFO 、套接字 、消息队列 、信号量 、共享存储对象

2. 类型信息存储在哪里？

文件类型信息存储在 struct stat 下的st_mode成员中的某一些位的编码

3. 如何提取类型信息？

利用系统提供的宏函数进行类型的提取, 返回谓词

S_ISREG(st_mode)

S_ISDIR(st_mode)

S_ISCHR(st_mode)

S_ISBLK(st_mode)

S_ISFIFO(st_mode)

S_ISLNK(st_mode)

S_ISSOCK(st_mode)

S_TYPEISMQ(struct stat *)

S_TYPEISSEM(struct stat *)

S_TYPEISSHM(struct stat *)

文件属性_设置用户ID和组ID

从进程方面来看，ID的分类

1.实际用户ID和组ID，表明我们的实际身份

2.有效用户ID和组ID， 通常，等于实际的用户和组ID，用来进行文件访问的权限检查

3.保存的设置用户ID和组ID， 在执行程序的时候，包含有效ID的副本

文件中st_mode中的设置组ID位和设置用户ID位：当执行该文件的时候，将进程的有效用户ID设置为文件的所有者的用户ID(st_uid)，设置组ID相同的作用

所有者ID是文件的性质， 有效ID是进程的性质

文件属性_文件访问权限

```
S_IRUSR //用户读
S_IWUSR
S_IXUSR

S_IRGRP
S_IWGRP
S_IXGRP

S_IROTH
S_IWOTH
S_IXOTH
```

1. 如果用路径打开某一个文件的时候，必须要对路径上的所有目录具有执行权限。

2. 目录的读权限和执行权限是不同的，读权限意味着我们可以度目录，获得在该目录中所有文件名的列表。执行权限允许我们搜索该目录，通过该目录，得到其下的一个文件的信息。目录的写权限意味着可以在目录下删除新建文件。

3. 对于文件的读写权限意味着我们是否可以对该文件进行读写操作。

4. exec执行某个文件，则该文件必须具有可执行权限，且是普通文件。

```
if(进程有效用户ID==0)
{
    允许访问
}
else if(进程的有效用户ID==文件的拥有者ID)
{
    查看对应权限位是否设置，设置就允许，没有设置就拒绝
}else if(进程的有效组ID==文件的所在组ID)
{
    查看对应权限位是否设置，设置就允许，没有设置就拒绝
}else
{
    查看其他人对应权限位是否设置，设置就允许，没有设置就拒绝
}
```

文件属性_创建的新文件和目录属于谁？

1. 新文件的用户ID设置为进程的有效用户ID

2. 新文件的组ID可以是进程的有效组ID，也可以是所在目录的组ID

文件属性_访问权限测试

有时候，我们迫切的想知道使用进程的实际ID对某个文件进行操作满不满足权限要求

int access(const char *pathname, int mode)

int faccessat(int fd, const char *pathname, int mode, int flag)

成功，返回0，失败，返回-1

```
mode =
R_OK     //是否可以读该文件
W_OK     //是否可以写该文件
X_OK    //是否可以执行该文件
F_OK   //文件是否存在

flag =
AT_EACCESS   //使用有效ID进行权限检查
```

文件属性_文件权限屏蔽字

mode_t umask(mode_t cmask)
设置新的权限屏蔽字，并且返回之前的屏蔽字

真正的权限 = 创建文件时设置的mode & ~umask

文件属性_更改文件的访问权限

int chmod(const char *pathname, mode_t mode);

int fchmod(int fd, mode_t mode);

int fchmodat(int fd, const char *pathname, mode_t mode, int flag);

成功，返回0，失败，返回-1

```
S_ISUID    //设置用户ID位
S_ISGID    //设置组ID位
S_ISVTX    //保存正文(粘着位)

S_IRWXU
    S_IRUSR
    S_IWUSR
    S_IXUSR
S_IRWXG
    S_IRGRP
    S_IWGRP
    S_IXGRP
S_IRWXO
    S_IROTH
    S_IWOTH
    S_IXOTH
```

1. 只有进程的有效用户ID等于文件的拥有者ID或者进程为超级用户，才可以修改文件的权限

2. flag设置为AT_STMLINK_NOFOLLOWt将不穿透符号链接

3. 对于fchmodat，如果pathname是相对目录，那么就会计算相对于fd的实际目录，如果pathname是绝对目录，那么fd将会被忽略。特别的，fd设置为AT_FDCWD, 代表参考位置是该进程的工作目录。

4.只有超级用户才可以设置保存正文位。其他用户就算设置了也会被自动清除。如果创建的文件所属组ID和创建其的进程的组ID或者附属组ID不一样，而且该进程不是超级用户，那么文件的设置组ID将会清除。

文件属性_粘着位

1. 最早的粘着位是针对于某些可执行程序设置的，如果被设置了该位，那么在程序执行完成后，会有一个副本被存在交换区中。使得下一次执行该程序能够快速的装入内存。

2.现阶段对目录设置粘着位，只有对该目录具有写权限且满足下面条件之一：

拥有此文件
拥有此目录
是超级用户

才可以删除或者重命名目录下的文件

文件操作_所有者更改

1. int chown(const char *pathname, uid_t owner, gid_t group)

2. int fchown(int fd, uid_t owner, gid_t group)

3. int fchownat(int fd, const char *pathname, uid_t owner, gid_t group, int flag)

4. int lchown(const char *pathname, uid_t owner, gid_t group)

成功，返回0，出错，返回-1

_POSIX_CHOWN_RESTRICTED: 如果针对于文件被设置：

1. 超级用户可以随意更改文件的用户ID
2. 如果某个进程拥有此文件（进程的有效用户ID等于文件的用户ID）， 参数owner为-1或者文件的用户ID（不能改成其他用户），参数group为进程的有效组ID或者附属组ID之一，则可以修改。即普通进程不能改其他用户文件的用户ID，但是可以更改自己拥有的文件的组ID，但是只能改到自己所属的组。

文件属性_文件长度

位于struct stat 结构体中的st_size, 存放在i节点信息中。

st_blksize: 对文件IO合适的块大小
st_blocks: 对文件所分配的实际的块数

1. 对于普通文件，文件的长度为实际存储内容的字节数
2. 对于符号链接，文件的长度为指向的文件路径的实际字节数

当我们将st_blksize 设置为一次读的操作字节数的时候，花费时间最少

文件属性_文件空洞

形成原因？

所设置的文件偏移量超过文件尾部，并写入某些数据造成

```
zhaosong@zhaosong:~/WorkSpace/APUE/4$ ls -l zero
-rw-r--r-- 1 zhaosong zhaosong 10003004 Aug 28 09:07 zero
zhaosong@zhaosong:~/WorkSpace/APUE/4$ du -s zero
8       zero
zhaosong@zhaosong:~/WorkSpace/APUE/4$ wc -c zero
10003004 zero
zhaosong@zhaosong:~/WorkSpace/APUE/4$ cat zero > zero.copy
zhaosong@zhaosong:~/WorkSpace/APUE/4$ ls -l zero*
-rw-r--r-- 1 zhaosong zhaosong 10003004 Aug 28 09:07 zero
-rw-r--r-- 1 zhaosong zhaosong 10003004 Aug 28 09:08 zero.copy
zhaosong@zhaosong:~/WorkSpace/APUE/4$ du -s zero*
8       zero
9772    zero.copy
```

ls : 列出文件占用的字节数,包含空洞

du：列出文件占用的文件块的个数，不包含空洞

正常的IO操作读取整个文件的长度，包含空洞，空洞的字节值为0.

对空洞文件进行复制，那么新的文件里面的空洞将会被赋予物理存储空间，且储值为0.

文件操作_文件截断

int truncate(const char *pathname, off_t length)

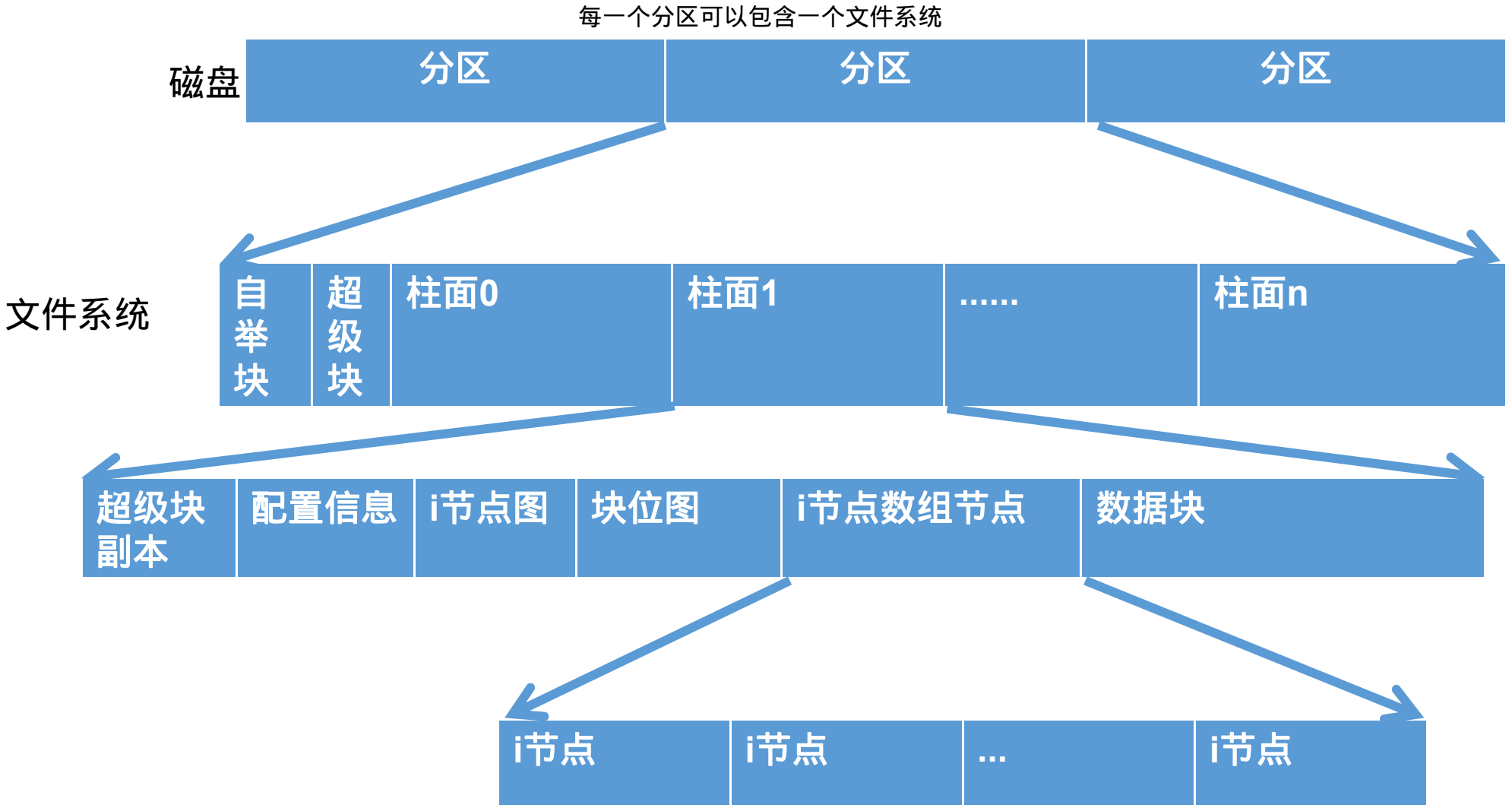
int ftruncate(int fd, off_t length)

成功，返回0，失败，返回-1

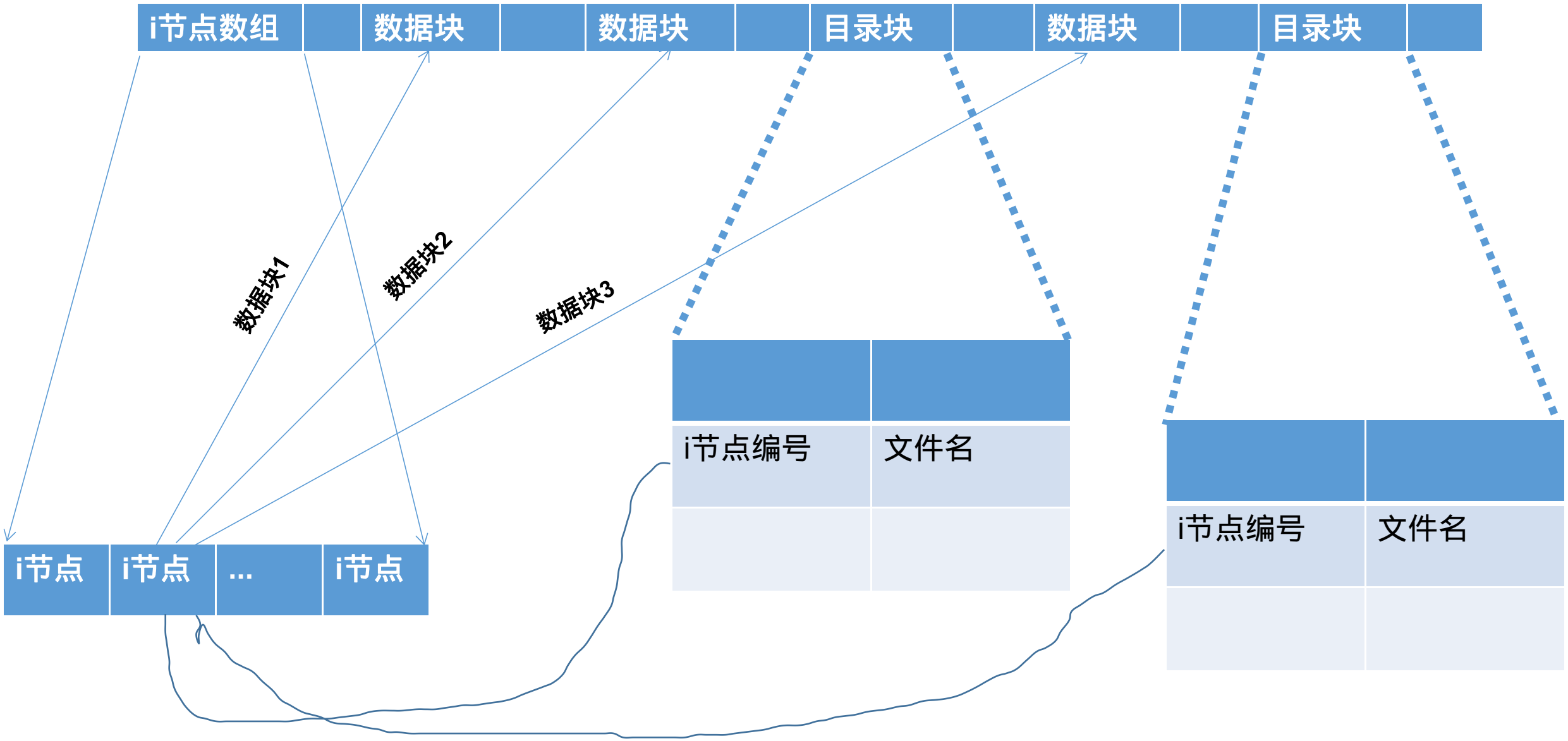
如果length比原来的文件长度小，那么多余的数据会被截断，不能再访问。

如果length大于文件长度，那么就会拓展文件位为空洞文件

文件系统结构


每一个分区可以包含一个文件系统
磁盘
分区
分区
分区
文件系统
自举块
超级块
柱面0
柱面1
.....
柱面n
超级块副本
配置信息
i节点图
块位图
i节点数组节点
数据块
i节点
i节点
...
i节点

文件系统结构


i节点数组
数据块
数据块
目录块
数据块
目录块
数据块1
数据块2
数据块3
i节点
i节点
...
i节点
i节点编号
文件名
i节点编号
文件名

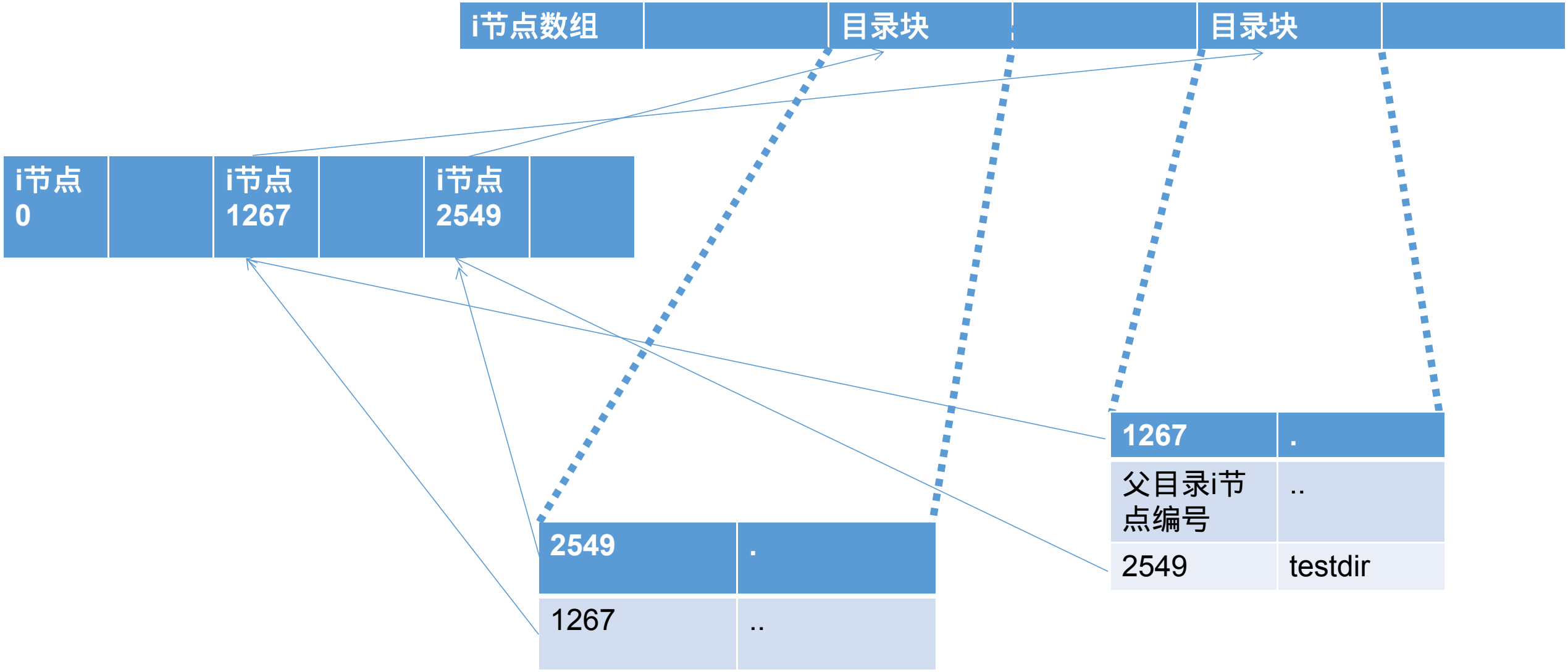
文件系统结构

1. i节点有一个属性是链接计数，在stat中是st_nlink. 连接计数就是指向该文件的目录项的个数。只有连接计数为0，才可以删除该文件，可以增加链接计数的链接称为硬链接。

2. 符号链接又称为软链接，存储的是实际指向的文件路径，不会增加所指向文件的链接计数

3. 目录项不可以指向另一个文件系统中的i节点

文件系统结构


i节点数组
目录块
目录块
i节点
0
i节点
1267
i节点
2549
1267
.
父目录i节点编号
..
2549
testdir
2549
.
1267
..

文件操作_创建硬链接

int link(const char *existingpath, const char *newpath)

int linkat(int efd, const char *existingpath, int nfd, const char *newpath, int flag)

成功返回0，失败返回-1

link不穿透符号链接，如果第一个参数指向一个符号链接文件，那么创建一个该符号链接的硬链接

linkat 可以通过设置flag参数位AT_SYMLINK_FOLLOW来穿透符号链接，如果标志被清除，则也不穿透符号链接

硬链接不允许跨文件系统

不允许创建一个指向目录的硬链接，有也仅限超级用户，因为会造成循环

int unlink(const char *pathname)

int unlinkat(int fd, const char *pathname, int flag)\

删除目录项，并将连接到的文件的链接数减1， 当flag参数设置为AT_REMOVEDIR时，unlinkat将类似于rmdir一样删除目录。
不穿透符号链接
成功返回0，失败返回-1

什么时候一个文件会被删除？ 打开该文件的进程数目为0，且文件的链接数目为0；两者必须同时满足

文件操作_remove

int remove(const char *pathname)

成功返回0，失败返回-1

解除对一个文件或者目录的链接，对于文件，remove的功能与unlink相同，对于目录，和rmdir相同

文件操作_更改名称

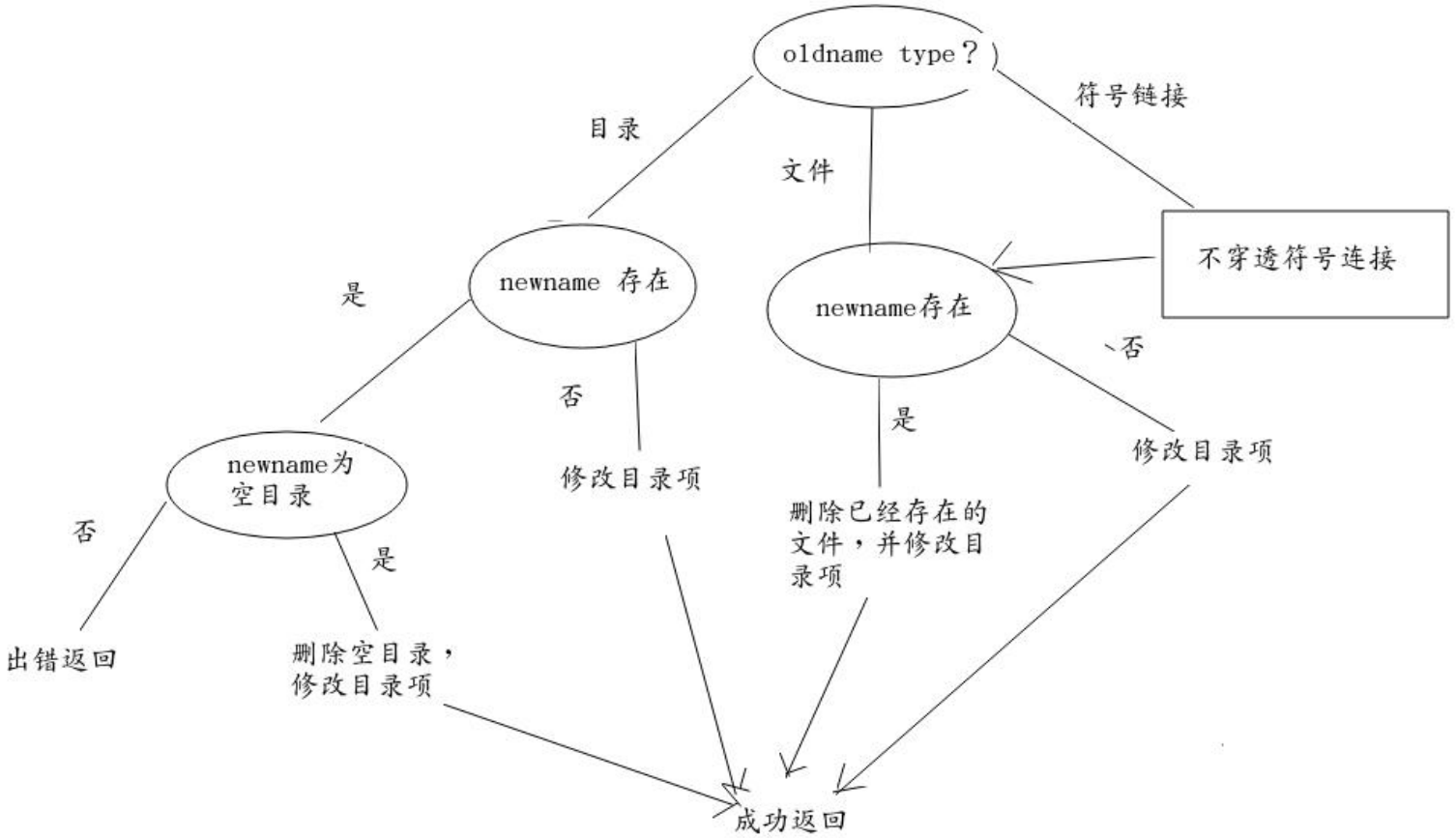
int rename(const char *oldname, const char *newname)

int renameat(int oldfd, const char *oldname, int newfd, const char *newname)

成功返回0，失败返回-1

oldname 是一个文件 、目录 、符号链接， newname已经存在或者不存在的处理方案？

文件操作_更改名称


oldname type？
目录
文件
符号链接
newname 存在
是
否
newname为
空目录
否
是
出错返回
删除空目录，
修改目录项
修改目录项
newname存在
是
否
不穿透符号连接
删除已经存在的
文件，并修改目
录项
修改目录项
成功返回

文件操作_符号链接相关

1. 符号链接和硬链接相比？
    硬链接只允许指向同一文件系统，符号链接可以随意
    硬链接不可以指向一个目录，符号链接可以
    硬链接增加文件的链接技术，符号链接不增加
2. 函数对符号链接的处理方案
    穿透符号链接
    不穿透符号链接
    参数为符号链接时，出错返回

3.创建和读取符号链接
    int symlink(const char *actualpath, const char *sympath)
    int symlinkat(const char *actualpath, int fd, const char *sympath)
成功，返回0，失败，返回-1

打开符号链接，并读取符号链接的内容（指向的文件的名字）

ssize_t readlink(const char *pathname, char *buf, size_t bufsize)

ssize_t readlinkat(int fd, const char *pathname, char *buf, size_t bufsize);

成功，返回读取的字节数，失败，返回-1

文件操作_文件的时间

st_atime : 文件数据的最后访问时间　ls -u

st_mtime：文件数据的最后修改时间 默认

st_ctime：i节点状态的最后修改时间　ls -c

```
struct timespec {
        time_t tv_sec;        /* seconds */
        long   tv_nsec;       /*nanoseconds */
    };
```

文件操作_文件的时间修改

```
struct timeval {
        long tv_sec;        /* seconds */
        long tv_usec;       /* microseconds */
    };
```

1. 修改文件内容的访问和修改时间

```
int futimens(int fd, const struct timespec times[2]);
int utimensat(int fd, const char *path, const struct timespec times[2], int flag);
int utimes(const char *pathname, const struct timeval times[2])
```

文件操作_目录的创建与删除

int mkdir(const char *pathname, mode_t mode)

int mkdirat(int fd, const char *pathname, mode_t mode)

成功，返回0，失败，返回-1

创建一个空的目录，其中. 和 ..自动创建，需要注意的是mode的可指向权限是应该被设置的

int rmdir(const char *pathname)

1. 删除一个空目录

2. 如果调用该函数时，没有其他进程占用此目录，那么直接释放，如果还有其他进程占用，则次函数会先删除文件中的.和.., 并把目录的链接计数设置为 0 ，并且，不可以在该文件中再创建新的文件，直到占有该目录的所有进程结束，才会真正的释放该目录。

文件操作_目录的读取

1. 用户只能读一个目录，但是只有内核可以写目录。

DIR *opendir(const char *pathname)

DIR *fdopendir(int fd)
成功，返回指针，失败，返回NULL

struct dirent *readdir(DIR *dp)
成功，返回指针，失败或者到达目录尾部，返回NULL

void rewinddir(DIR *dp)

int closedir(DIR *dp)
成功，返回 0 ，失败，返回-1

long telldir(DIR *dp)
返回与 d p 关联的目录中的当前位置

void seekdir(DIR *dp, long loc)

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

文件操作_工作目录相关

int chdir(const char *pathname)

int fchdir(int fd)

更改当前的工作目录

成功，返回 0 ，失败，返回-1

char *getcwd(char *buf, size_t size);

获取当前的工作目录

成功返回buff, 失败返回NULL

文件属性_设备字段

st_dev: 文件系统的设备号

st_rdev:特殊设备号，只有字符文件和块文件才具有

```c
#include<apue.h>
#include<sys/sysmacros.h>
int main()
{
    struct stat mystat;

    stat("./4_7.c", &mystat);

    printf("dev=%d/%d\n", major(mystat.st_dev), minor(mystat.st_dev));
    return 0;
}
```

标识设备驱动程序

标识特定的子设备