

第五章 文件与目录

1. 文件I/O

- 大多数UNIX文件I/O操作只需用到5个函数：

open、read、write、lseek、close

- 上述五个函数经常被称为不带缓存的I/O；
- 不带缓存指的是每个read和write都调用内核中的一个系统调用；
- 这些不带缓存的I/O函数不是ANSI C的组成部分，但是是POSIX.1和XPG3的组成部分。

1. 文件I/O

Wuhan University

- 1. 文件描述符

- 对与内核而言，所有打开文件都由文件描述符引用，文件描述符是一个非负整数；
- 当打开一个现存文件或创建一个新文件时，内核向进程返回一个文件描述符；
- 当读、写一个文件时，用open或create返回的文件描述符标识该文件；
- 文件描述符的范围是0~OPEN_MAX（通常为63）。

1. 文件I/O

- **open**函数

调用open函数可以打开或创建一个文件。

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

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

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

O_RDONLY 只读打开
O_WRONLY 只写打开
O_RDWR 读、写打开
.....
O_CREAT 若文件不存在则新建

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

打开或创建
的文件名

若oflag为O_CREAT，则需要第三个参数来设定新建文件权限 (0644)

1. 文件I/O

- **open**函数

*oflag*参数：

O_RDONLY、O_WRONLY、O_RDWR必须指定一个，而下列常数是可选的：

- O_APPEND 每次写时都追加到文件尾端
- O_CREAT 若文件不存在则创建它
- O_EXCL 如果同时指定O_CREAT，而文件已经存在，
则出错，可测试一个文件是否已经存在
- O_TRUNC 如果文件存在，而且为只读或只写成功打开，则将其长度截短为0

1. 文件I/O

Wuhan University

- **open**函数

*oflag*参数：

- **O_NOCTTY** 如果pathname指的是终端设备，则不将此设备分配作为此进程的控制终端
- **O_NONBLOCK** 如果pathname指的是一个FIFO、一个块特殊文件或一个字符特殊文件，则此选项为此文件的本次打开操作和后续的I/O操作设置非阻塞模式
- **O_SYNC** 使每次write都等到物理I/O操作完成

1. 文件I/O

- **create**函数

调用**create**函数可以创建一个新文件。

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

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

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

creat()函数的缺点是只能以只写方式打开，不能读取

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



等效于

```
open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
```

1. 文件I/O

- **close**函数

调用close函数可以关闭一个打开文件。

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

```
int close(int filedes);
```

- 关闭一个文件时也释放该进程加在该文件上的所有记录锁;
- 当一个进程终止时, 它所有的打开文件都由内核自动关闭。很多程序都使用这一功能而不显式地用close()关闭打开的文件。

1. 文件I/O

- **lseek**函数

调用lseek函数可以显示的定位一个打开文件。

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

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

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

whence == SEEK_SET 将文件的位移量设置为文件开始处*offset*个字节;

whence == SEEK_CUR 将文件的位移量设置为当前值加*offset*个字节;

whence == SEEK_END 将文件的位移量设置为文件长度加*offset*个字节;

若lseek()成功执行，则返回新的文件位移量，失败则返回-1

1. 文件I/O

- **lseek**函数

\$a.out < /etc/motd

\$Seek OK

\$a.out < /var/spool/cron/FIFO

\$Can not seek

- **lseek()**仅将当前的文件位移量记录在内核中，但并不引起任何I/O操作；
- 文件位移量可以大于文件的当前长度。

```
/* lseek.c */
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char **argv){
    if(lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
        printf("Can not seek\n");
    else
        printf("Seek OK\n");
    exit(0);
}
```

不能使用 < 0

1. 文件I/O

- **read**函数

调用read函数可以从一个文件中读数据。

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

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

不一定等于
read()返回值

- 如果read()成功，则返回读到的字节数；
- 如果已经到达文件末尾，则返回0；
- 从终端设备读时，通常一次最多读一行；
- 从网络读时，网络中的缓冲机制可能造成返回值小于所要求的读取值；

1. 文件I/O

- **write**函数

调用**write**函数向打开的文件中写数据。

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

通常等于write()
返回值

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

- 对于普通文件，写操作从文件的当前位移量出开始；
- 如果在打开文件时，指定了**O_APPEND**选项，则每次写操作之前，将文件位移量设置在文件的当前结尾处；
- 在一次成功写操作后，文件位移量增加实际写的字节数。

1. 文件I/O

Wuhan University

```
#include "apue.h"
#include <fcntl.h>

char    buf1[] = "abcdefghij";
char    buf2[] = "ABCDEFGHIJ";
int main(void){
    int    fd;
    if ((fd = creat("file.hole", FILE_MODE)) < 0)
        err_sys("creat error");
    if (write(fd, buf1, 10) != 10)           /* offset now = 10 */
        err_sys("buf1 write error");
    if (lseek(fd, 16384, SEEK_SET) == -1)    /* offset now = 16384 */
        err_sys("lseek error");
    if (write(fd, buf2, 10) != 10)
        err_sys("buf2 write error");      /* offset now = 16394 */
    exit(0);
}
```

1. 文件I/O

- **dup、dup2**函数

dup、dup2函数都可以用来复制一个现存的文件描述符。

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

```
int dup(int filedes);
```

返回值为当前
可以用最小的
文件描述符

```
int dup2(int filedes, int filedes2);
```

指定新的文件
描述符

若filedes2已经存在，则先关闭；若filedes等于filedes2，则直接返回filedes2，不关闭它

- 函数成功则返回新的文件描述符，若出错则返回-1；

2. 文件和目录

Wuhan University

- **stat、fstat、lstat**函数

三个函数可以用来获取文件的信息。

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

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

使用该函数最多的命令ls -l

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

```
int fstat(int fildes, struct stat *buf);
```

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

成功则返回0，失败则返回-1

2. 文件和目录

- **stat、fstat、lstat**函数

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

- 给定一个*pathname*，函数返回一个与此文件有关的信息结构；

int fstat(int *filedes*, struct stat **buf*);

- 给定一个文件描述符*filedes*，函数返回一个与此文件有关的信息结构；

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

- 同stat()，但是如果*pathname*是链接文件，则返回链接文件的信息结构；

2. 文件和目录

- **stat、fstat、lstat**函数

```
struct stat {  
    mode_t      st_mode;      /* file type & mode (permissions) */  
    ino_t       st_ino;       /* i-node number (serial number) */  
    dev_t       st_dev;       /* device number (file system) */  
    dev_t       st_rdev;      /* device number for special files */  
    nlink_t     st_nlink;     /* number of links */  
    uid_t       st_uid;       /* user ID of owner */  
    gid_t       st_gid;       /* group ID of owner */  
    off_t       st_size;      /* size in bytes, for regular files */  
    time_t      st_atime;     /* time of last access */  
    time_t      st_mtime;     /* time of last modification */  
    time_t      st_ctime;     /* time of last file status change */  
    blksize_t   st_blksize;   /* best I/O block size */  
    blkcnt_t    st_blocks;    /* number of disk blocks allocated */  
};
```


2. 文件和目录

- 文件类型

- **普通文件**(regular file) : 最常见的文件类型, 包含了某种格式的数据 ;
- **目录文件**(directory file) : 包含了其他文件的名字以及指向这些文件有关信息的指针; 对一个目录文件有读许可权的进程都可以读该目录的内容, 但只有内核可以写目录;
- **字符特殊文件**(character special file) : 用于系统中某些类型的设备 ;
- **块特殊文件**(block special file) : 磁盘设备。系统中的所有设备不是字符特殊文件就是块特殊文件;
- **FIFO** : 用于进程间通信的文件;
- **套接口**(socket) : 用于进程间的网络通信;
- **符号链接**(symbolic link) : 这种文件指向另一个文件。

2. 文件和目录

[例子]：列出当前目录下的所有文件

ls -al

```
[gene@localhost ~]$ ls -al
total 54280
drwx----- 4 gene gene      4096 2008-08-21 22:39 .
drwxr-xr-x  3 root root      4096 2008-08-11 10:59 ..
-rw-----  1 gene gene        338 2008-08-21 00:58 .bash_history
-rw-r--r--  1 gene gene         33 2008-08-11 10:59 .bash_logout
-rw-r--r--  1 gene gene        176 2008-08-11 10:59 .bash_profile
-rw-r--r--  1 gene gene        434 2008-08-11 11:14 .bashrc
drwxr-xr-x  2 gene gene      4096 2008-08-11 10:59 .gnome2
drwxr-xr-x 18 502 wheel      4096 2008-08-11 11:06 ns-allinone-2.33
-rwxr-xr-x  1 root root 55428314 2008-08-11 11:00 ns-allinone-2.33.tar.gz
lrwxrwxrwx  1 gene gene         8 2008-08-21 22:39 shortcut -> .viminfo
```

[说明]：- 为普通文件

d 为目录

l 为链接

b 为块文件

c 为字符型文件

p 为命名管道（FIFO）

2. 文件和目录

- 文件类型

- 文件类型信息包含在stat结构的st_mode成员中，可以通过宏来确定文件类型。

<sys/stat.h>中的文件类型宏	
宏	文件类型
S_ISREG()	普通文件
S_ISDIR()	目录文件
S_ISCHR()	字符特殊文件
S_ISBLK()	块特殊文件
S_ISFIFO()	管道文件
S_ISLNK()	符号链接文件
S_ISSOCK()	套接字


```

#include    <sys/types.h>
#include    <sys/stat.h>
#include    "ourhdr.h"

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;
        }

        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";
#ifdef S_ISLNK
        else if (S_ISLNK(buf.st_mode)) ptr = "symbolic link";
#endif
#ifdef S_ISSOCK
        else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
#endif
        else
            ptr = "*** unknown mode ***";
        printf("%s\n", ptr);
    }
    exit(0);
}

```

2. 文件和目录

- **access**函数

按实际用户和组进行存取许可权测试

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

```
int access(const char *pathname, int mode);
```

Mode	说明
R_OK	测试读许可权
W_OK	测试写许可权
X_OK	测试执行许可权
F_OK	测试文件是否存在

```
#include    <sys/types.h>
#include    <fcntl.h>
#include    "ourhdr.h"

int
main(int argc, char *argv[])
{
    if (argc != 2)
        err_quit("usage: a.out <pathname>");

    if (access(argv[1], R_OK) < 0)
        err_ret("access error for %s", argv[1]);
    else
        printf("read access OK\n");

    if (open(argv[1], O_RDONLY) < 0)
        err_ret("open error for %s", argv[1]);
    else
        printf("open for reading OK\n");

    exit(0);
}
```


2. 文件和目录

- **umask**函数




在进程创建一个新文件或新目录时，使用屏蔽字

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

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

```
mode_t umask(mode_t cmask);
```

通过四个八进制数直接赋值；

	<u>r</u> <u>w</u> <u>x</u>	<u>r</u> <u>w</u> <u>x</u>	<u>r</u> <u>w</u> <u>x</u>
—	1 1 1	1 1 1	1 1 1
	0 0 0	0 0 0	0 0 0
			
	0 ~ 7	0 ~ 7	0 ~ 7

```

#include    <sys/types.h>
#include    <sys/stat.h>
#include    <fcntl.h>
#include    "ourhdr.h"

int
main(void)
{
    umask(0);
    if (creat("foo", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
               S_IROTH | S_IWOTH) < 0)
        err_sys("creat error for foo");

    umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
    if (creat("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
               S_IROTH | S_IWOTH) < 0)
        err_sys("creat error for bar");
    exit(0);
}

```

\$ a.out

4 ls -l foo bar

```

-rw----- 1 stevens      0 Nov 16 16:23 bar
-rw-rw-rw- 1 stevens      0 Nov 16 16:23 foo

```

2. 文件和目录

Wuhan University

- **chmod、fchmod**函数

这两个函数可以更改现存文件的存取许可权

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

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

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

```
int fchmod(int fildes, mode_t mode);
```

- 若成功则返回0，失败则返回-1

<i>mode</i>	说明
S_IRWXU	所有者读、写、执行
S_IRUSR	所有者读
S_IWUSR	所有者写
S_IXUSR	所有者执行
S_IRWXG	组读、写、执行
S_IRGRP	组读
S_IWGRP	组写
S_IXGRP	组执行
S_IRWXO	其他读、写、执行
S_IROTH	其他读
S_IWOTH	其他写
S_IXOTH	其他执行

```

#include    <sys/types.h>
#include    <sys/stat.h>
#include    "ourhdr.h"

int
main(void)
{
    struct stat    statbuf;

    /* turn on set-group-ID and turn off group-execute */

    if (stat("foo", &statbuf) < 0)
        err_sys("stat error for foo");
    if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
        err_sys("chmod error for foo");

    /* set absolute mode to "rw-r--r--" */

    if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
        err_sys("chmod error for bar");

    exit(0);
}

```

2. 文件和目录

Wuhan University

- **chown、fchown、lchown**函数
更改文件的用户ID和组ID

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

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

```
int chown(const char *pathname, uid_t owner, gid_t group);
```

```
int fchown(int fildes, uid_t owner, gid_t group);
```

```
int lchown(const char *pathname, uid_t owner, gid_t group);
```

- 若成功则返回0，失败则返回-1

2. 文件和目录

- 文件系统

引导块	超级块	i 节 点 区	文 件 存 储 区
-----	-----	---------	-----------

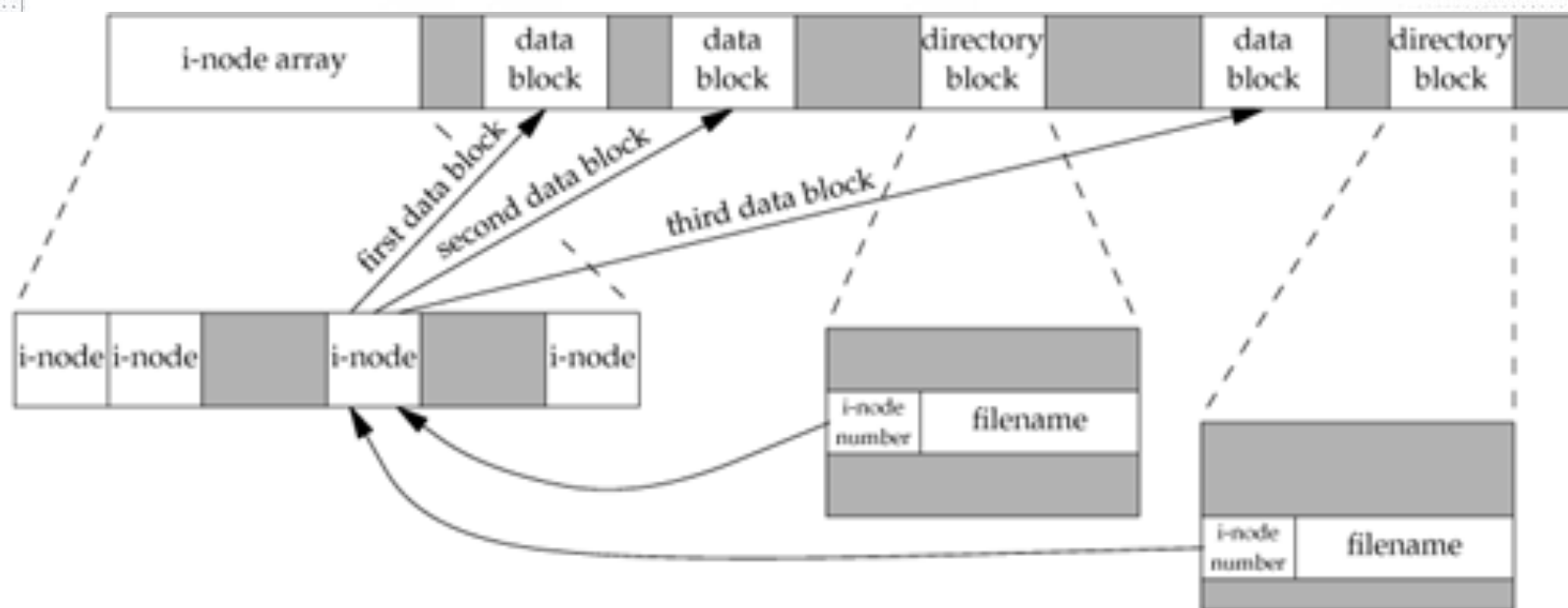
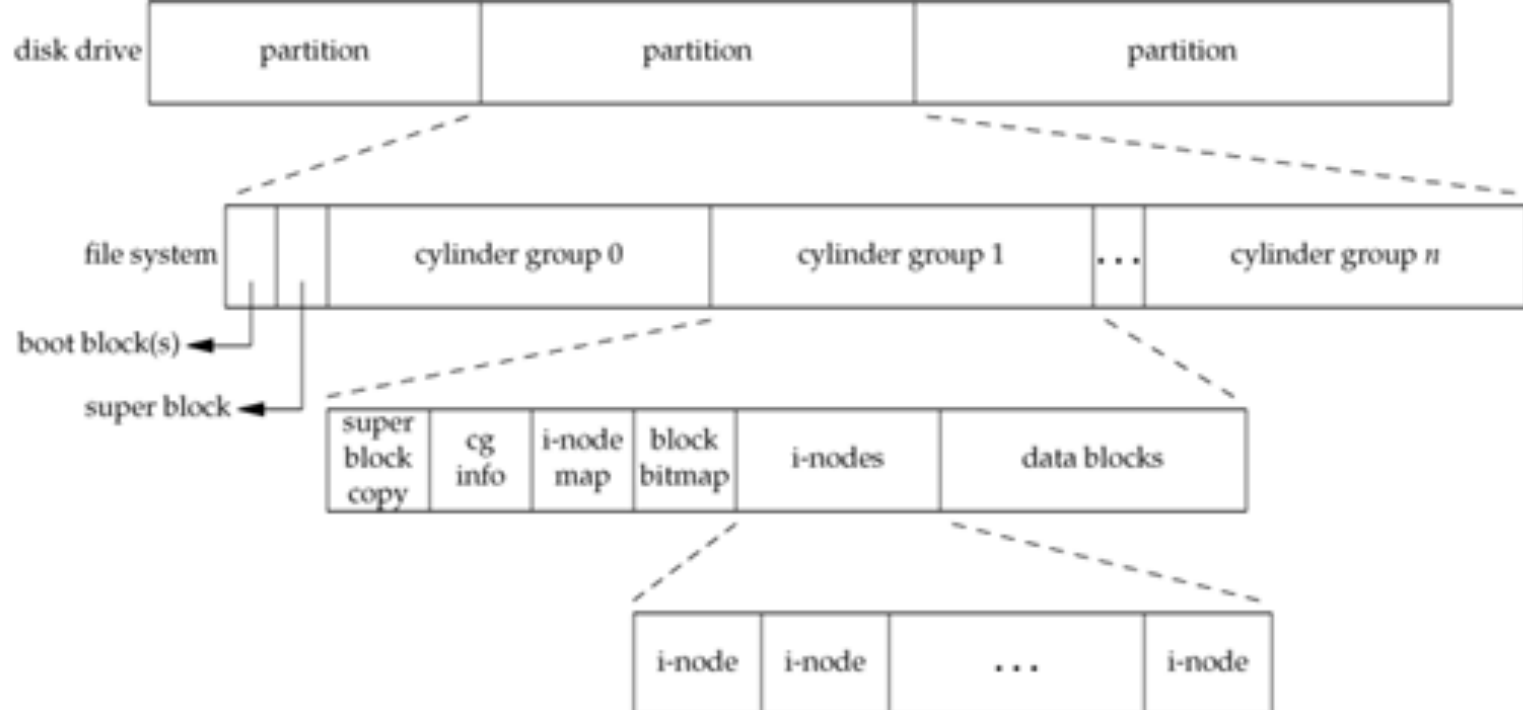
- **引导块 (Boot block)** : 0号块。用于启动系统, 存放引导程序, 它含有的程序代码用于系统启动时引导执行操作系统的内核;
- **超级块 (Super block)** : 1号块, 也叫管理块。存放与整个文件系统的管理有关的信息;

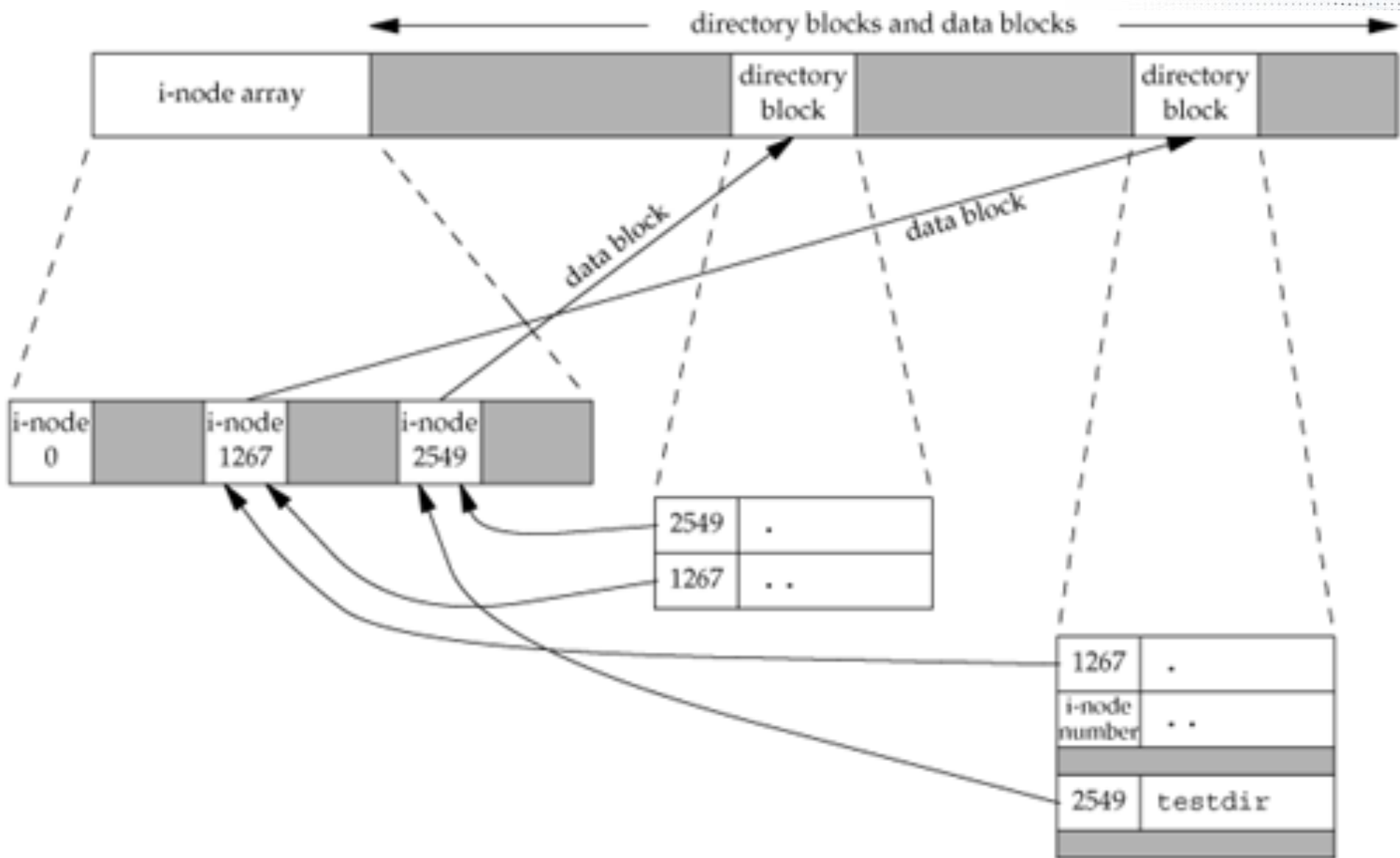
2. 文件和目录

Wuhan University

- 文件系统

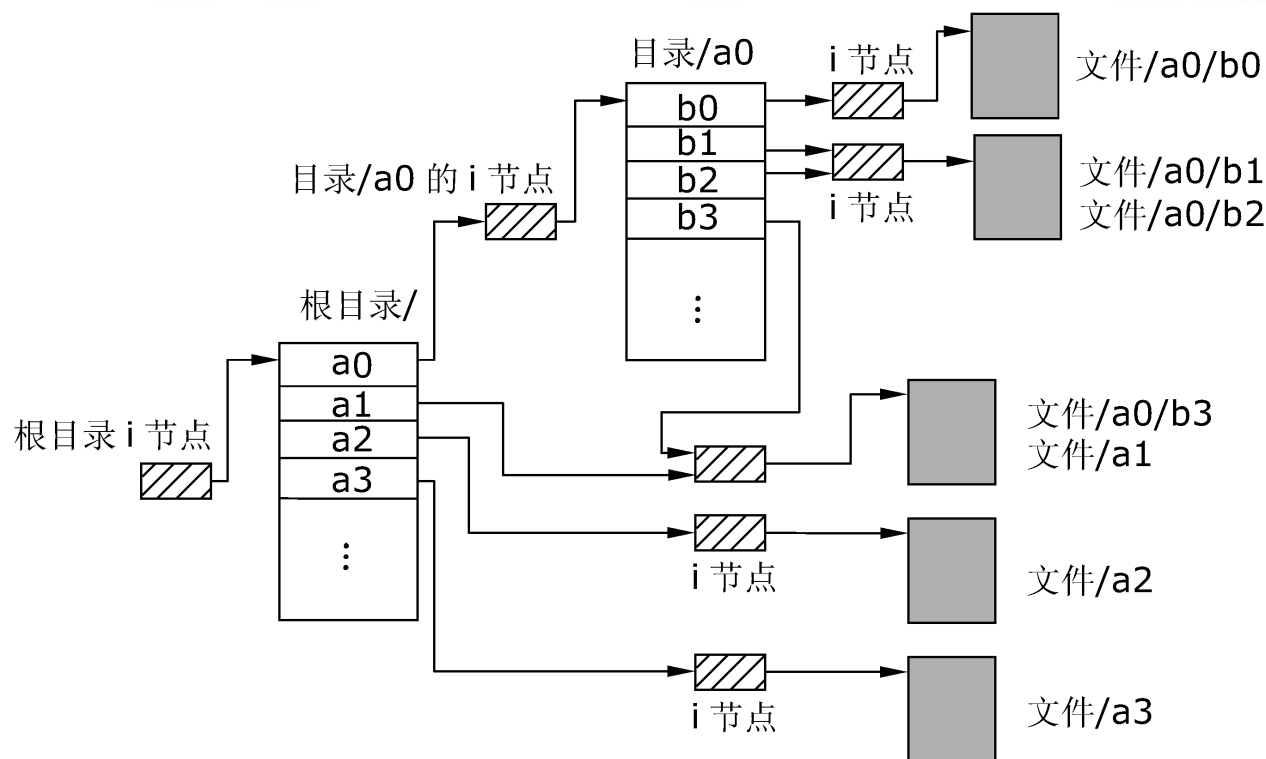
- **i节点区**：i节点(index node)，简记为i-node。i节点区由若干块构成，专用于存放i节点。系统中的每个文件都对应一个i节点。每块可容多个i节点，每个i节点有固定大小。i节点中最重要的信息是“索引”信息。i节点中还记录了一些文件属性信息。注意：i节点内不含有文件的文件名。i节点编号从1开始。1, 2, 3, ..., 不使用编号为0的i节点。





2. 文件和目录

- 目录



- 一个文件可以有多个名字。只要在不同的目录项中填写“文件名-i节点号”的时候将i节点号填写的相同就可以了；

2. 文件和目录

- **link、unlink**函数

link()为创建硬链接函数，unlink()为删除硬链接函数；

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

成功则硬链接
计数加1

```
int link(const char *pathname, const char *newpass);
```

```
int unlink(const char *pathname);
```

成功则硬链接
计数减1

- 只有当硬链接计数器为0时，该文件才能被删除。

2. 文件和目录

- **symlink**函数

symlink函数创建一个符号链接;

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

```
int symlink(const char *pathname, const char *newpass);
```

符号链接是对一个文件的间接指针，它与硬链接不同，硬链接直接指向文件的i-node，而符号链接以保存目标文件路径的形式链接到目标文件。

2. 文件和目录

- **utime**函数

文件的存取和修改时间可以用**utime()**来实现;

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

```
#include <utime.h>
```

```
int utime(const char *pathname, const struct utimbuf *times);
```

```
struct utimbuf{
```

```
    time_t actime;           /* access time */
```

```
    time_t modtime;         /* modification time */
```

```
}
```

```

#include "apue.h"
#include <fcntl.h>
#include <utime.h>

int
main(int argc, char *argv[])
{
    int          i, fd;
    struct stat   statbuf;
    struct utimbuf timebuf;

    for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) { /* fetch current times */
            err_ret("%s: stat error", argv[i]);
            continue;
        }
        if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
            err_ret("%s: open error", argv[i]);
            continue;
        }
        close(fd);
        timebuf.actime = statbuf.st_atime;
        timebuf.modtime = statbuf.st_mtime;
        if (utime(argv[i], &timebuf) < 0) { /* reset times */
            err_ret("%s: utime error", argv[i]);
            continue;
        }
    }
    exit(0);
}

```

2. 文件和目录

Wuhan University

- mkdir、rmdir函数

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

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

```
int mkdir(const char *pathname, mode_t *mode);
```

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

必须为空目录

2. 文件和目录

- 读目录

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

```
#include <dirent.h>
```

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

```
struct dirent *readdir(DIR *dp);
```

```
void rewinddir(DIR *dp)
```

```
int closedir(DIR *dp)
```



```
struct dirent {  
    ino_t d_ino; /* i-node number */  
    char d_name(NAME_MAX+1);  
}
```


2. 文件和目录

Wuhan University

- **chdir**、**fchdir**函数

两函数用于更改当前工作目录;

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

```
int chdir(const char *pathname);
```

```
int fchdir(int filedes);
```

```
#include "ourhdr.h"
```

```
Int main(void)
```

```
{
```

```
    if(chdir("/tmp") < 0)
```

```
        err_sys("chdir failed");
```

```
    printf("chdir to /tmp succeeded\n");
```

```
    exit(0);
```

```
}
```

2. 文件和目录

- **getcwd**函数

用于获取当前工作目录的绝对路径；

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

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

缓存地址，返回当前
工作目录的绝对路径

缓存大小，必须有足够长度容
纳路径名和一个null终止符

2. 文件和目录

- **sync、fsync**函数

同步磁盘文件系统与缓存中内容的一致性;

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

```
void *sync(void);
```

```
int fsync(int fildes);
```

- **sync()**只是将所有修改过的块的缓存排入写操作，然后就返回，它并不等待实际I/O操作结束;
- **fsync()**只引用单个文件，等待I/O结束，然后返回；**fsync()**可用于数据这样的应用程序，确保修改过的块立即写到磁盘上;