



Ch3 Linux System Programming

– File System

Enyi Tang
Software Institute of
Nanjing University



What is File and File System?

- File

- An object that can be written to, or read from, or both. A file has certain attributes, including access permissions and type.

- File system

- A collection of files and certain of their attributes. It provides a name space for file serial numbers referring to those files.



文件系统的多种含义

■ 文件系统

■ Linux内核源代码情景分析，P415

- (1) 指一种特定的文件格式。例如，我们说 Linux 的文件系统是 Ext2，MSDOS 的文件系统是 FAT16，而 Windows NT 的文件系统是 NTFS 或 FAT32，就是指这个意思。
- (2) 指按特定格式进行了“格式化”的一块存储介质。当我们说“安装”或“拆卸”一个文件系统时，指的就是这个意思。
- (3) 指操作系统中（通常在内核中）用来管理文件系统以及对文件进行操作的机制及其实现，这就是本章的主要话题。

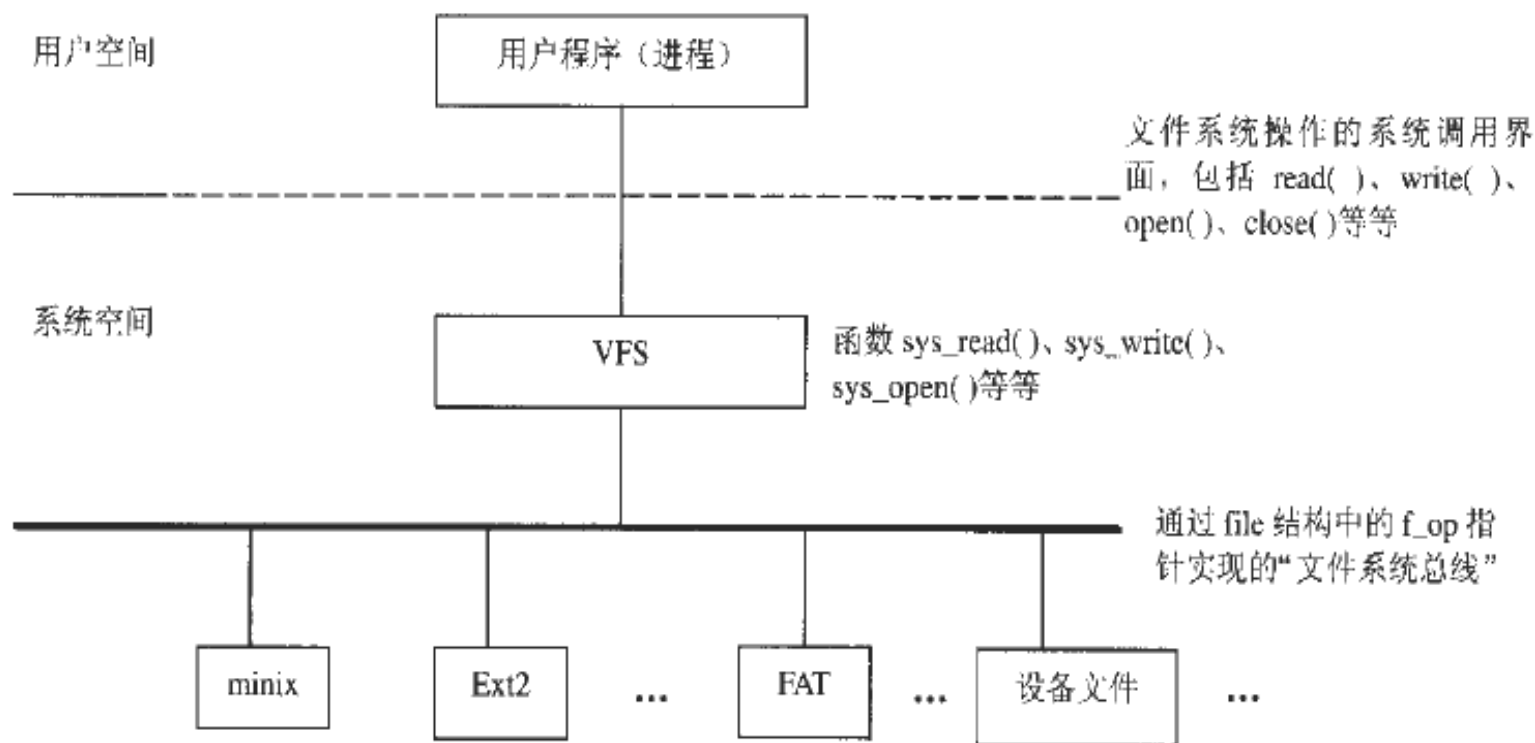


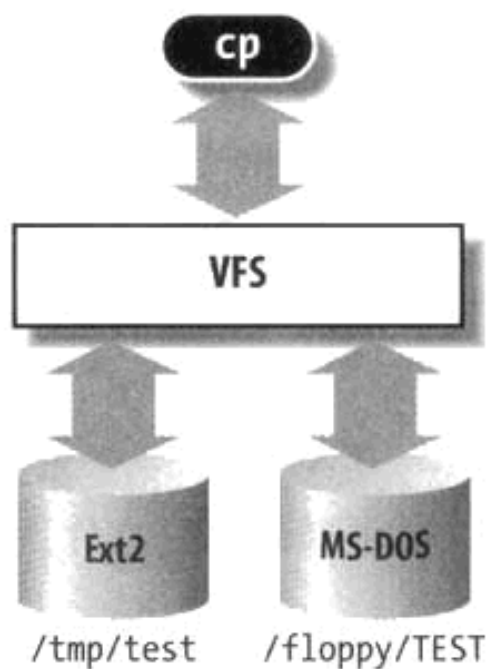
File Types and Structure

- File types
 - regular file
 - character special file
 - block special file
 - fifo
 - socket
 - symbolic link
 - directory
- File structure
 - Byte stream; no particular internal structure

File Systems in Linux

■ Virtual File system Switch (VFS)





(a)

```
inf = open("/floppy/TEST", O_RDONLY, 0);  
outf = open("/tmp/test",  
            O_WRONLY|O_CREAT|O_TRUNC, 0600);  
do {  
    i = read(inf, buf, 4096);  
    write(outf, buf, i);  
} while (i);  
close(outf);  
close(inf);
```

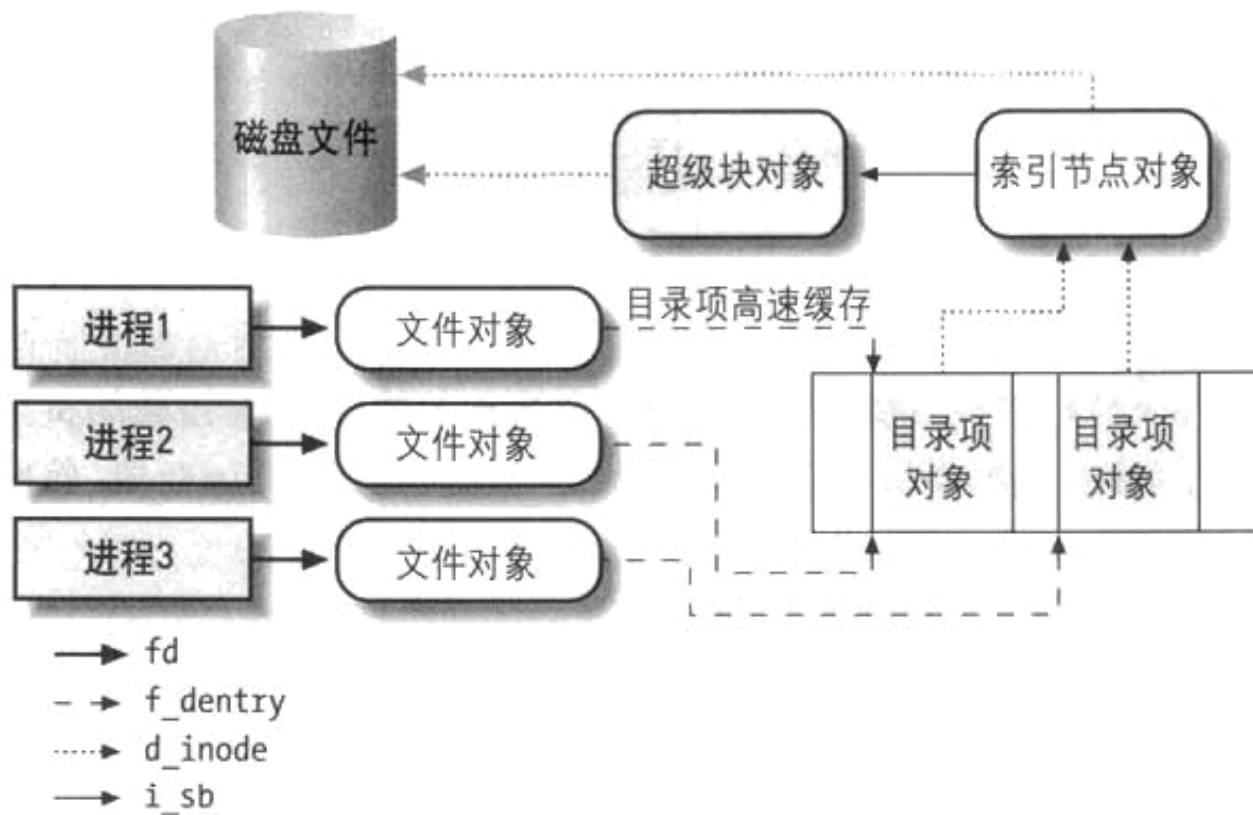
(b)

图 12-1: VFS 在一个简单的文件复制操作中的作用



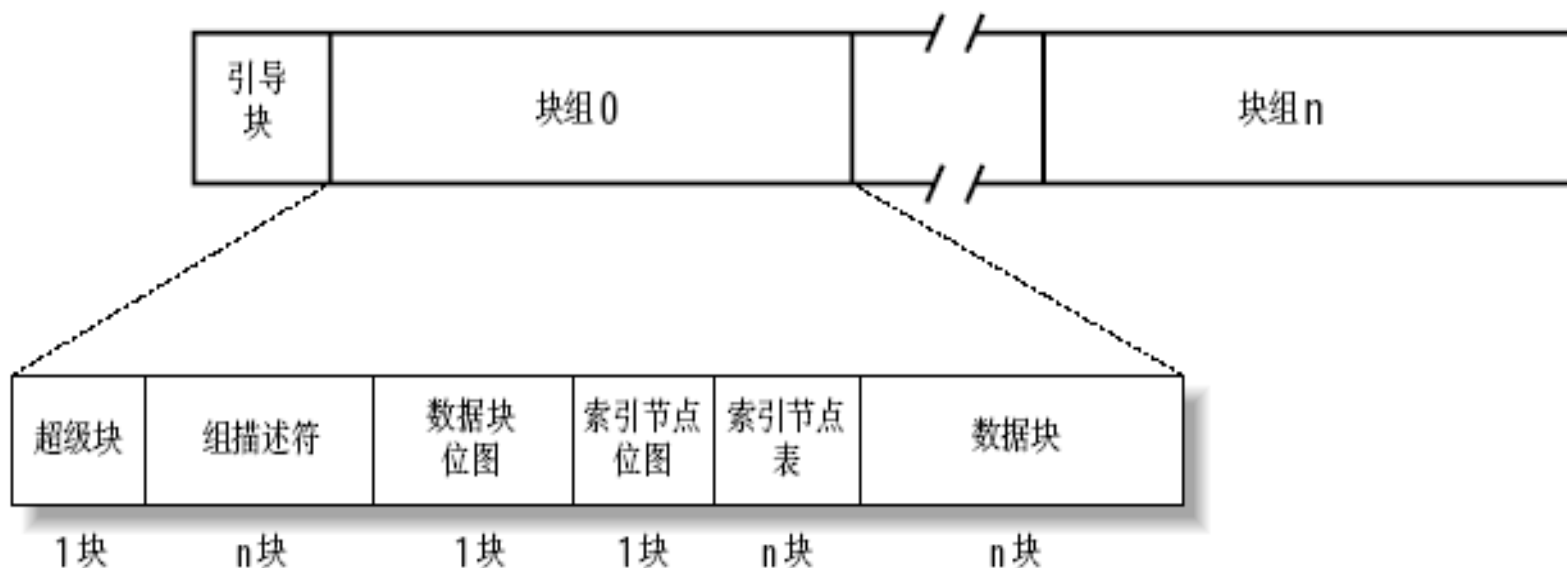
VFS Model

- Virtual; only exists in memory
- Components:
 - super block
 - i-node object
 - file object
 - dentry object



Ext2 File System

■ Ext2 Architecture





Hard link and symbolic link

- Hard link

- 不同的文件名对应同一个inode
- 不能跨越文件系统
- 对应系统调用link

- Symbolic link

- 存储被链接文件的文件名(而不是inode)实现链接
- 可跨越文件系统
- 对应系统调用symlink

Review "ls -l"

To show the permissions of a file, use the **ls** command with the **-l** option

```
$ ls -l
```

-rw-r--r--	1	tux1	penguins	101	Jun 5 10:03	file1
-rw-r--r--	1	tux2	penguins	171	Jun 4 10:23	file2
drwxr-xr-x	2	tux1	penguins	512	Jun 7 11:13	mydir

File type

permissions

link counter

owner

group

size

mtime
(modification time)

name



System Calls & Library Functions

- 都以C函数的形式出现
- 系统调用
 - Linux内核的对外接口; 用户程序和内核之间唯一的接口; 提供最小接口
- 库函数
 - 依赖于系统调用; 提供较复杂功能
 - 例: 标准I/O库



Unbuffered I/O & Buffered I/O

- Unbuffered I/O
 - read/write -> System calls
 - File descriptor
 - Not in ANSI C, but in POSIX.1 and XPG3
- Buffered I/O
 - Implemented in standard I/O library
 - 处理很多细节, 如缓存分配, 以优化长度执行I/O等.
 - Stream -> a pointer to FILE



Basic I/O System Calls

- File descriptor
- Basic I/O
 - open/creat, close, read, write, lseek
 - dup/dup2
 - fcntl
 - ioctl



File Descriptor

- File descriptor
 - A small non-negative integer
 - `int fd;`
 - (in `<unistd.h>`)
 - `STDIN_FILENO` (0), `STDOUT_FILENO` (1), `STDERR_FILENO` (2)
- General steps of file operation
 - open-read/write-[lseek]-close



Example

```
/* a rudimentary example program */  
#include <fcntl.h>
```

```
main()  
{  
    int fd, nread;  
    char buf[1024];  
  
    /*open file "data" for reading */  
    fd = open("data", O_RDONLY);  
  
    /* read in the data */  
    nread = read(fd, buf, 1024);  
  
    /* close the file */  
    close(fd);  
}
```




open/creat Function

- Open and possibly create a file or device

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

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

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

```
int open(const char *pathname, int flags);
```

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

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

(Return: a new file descriptor if success; -1 if failure)



Parameter “flags”

- “flags”: file access mode
 - One of **O_RDONLY**, **O_WRONLY** or **O_RDWR** which request opening the file read-only, write-only or read/write, respectively, bitwise-or'd with zero or more of the following:
(All defined in /usr/include/fcntl.h)
 - O_APPEND**: the file is opened in append mode
 - O_TRUNC**: If the file already exists and is a regular file and the open mode allows writing will be truncated to length 0.
 - O_CREAT**: If the file does not exist it will be created.
 - O_EXCL**: When used with **O_CREAT**, if the file already exists it is an error and the open will fail.
 - ...
- “creat” function: equivalent to open with flags equal to **O_CREAT|O_WRONLY|O_TRUNC**



Parameter “mode”

- “mode”: specifies the permissions to use in case a new file is created.



The value of parameter “mode”

取值	含义
S_IRUSR(00400)	Read by owner
S_IWUSR(00200)	Write by owner
S_IXUSR(00100)	Execute by owner
S_IRWXU(00700)	Read, write and execute by owner
S_IRGRP 00040	Read by group
S_IWGRP 00020	Write by group
S_IXGRP 00010	Execute by group
S_IRWXG 00070	Read, write and execute by group
S_IROTH 00004	Read by others
S_IWOTH 00002	Write by others
S_IXOTH 00001	Execute by others
S_IRWXO 00007	Read, write and execute by others



Parameter "mode" & umask

- umask: a file protection mechanism
- The initial access mode of a new file
 - $\text{mode} \& \sim \text{umask}$

regular files:

default permissions	rw-rw-rw-	666
umask (-)	---W--W-	022
resulting permissions	rw-r--r--	644

directories:

default permissions	rw-rwxrwx	777
umask (-)	---W--W-	022
resulting permissions	rw-r-xr-x	755



close Function

- Close a file descriptor

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

```
int close(int fd);
```

(Return: 0 if success; -1 if failure)



read/write Function

- Read from a file descriptor

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

```
ssize_t read(int fd, void *buf, size_t count);
```

(返回值: 读到的字节数, 若已到文件尾为0, 若出错为-1)

- Write to a file descriptor

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

```
ssize_t write(int fd, const void *buf, size_t count);
```

(返回值: 若成功为已写的字节数, 若出错为-1)



Example

- **mycat.c**
- ```
while ((n = read(STDIN_FILENO, buf, BUFSIZE)) > 0)
 if (write(STDOUT_FILENO, buf, n) != n)
 err_sys("write error");
 if (n < 0)
 err_sys("read error");
```





# *lseek* Function

---

- Reposition read/write file offset

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

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

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

(Return: the resulting offset location if success; -1 if failure)

- The directive "whence":

- **SEEK\_SET**: the offset is set to "offset" bytes

- **SEEK\_CUR**: the offset is set to its current location plus "offset" bytes

- **SEEK\_END**: the offset is set to the size of the file plus "offset" bytes



# *dup/dup2* Function

---

- Duplicate a file descriptor
  - #include <unistd.h>
  - int dup(int oldfd);
  - int dup2(int oldfd, int newfd);
  - (Return: the new file descriptor if success; -1 if failure)
- File sharing
  - Example: redirection



# *fcntl* Function

---

- Manipulate a file descriptor

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

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

```
int fcntl(int fd, int cmd);
```

```
int fcntl(int fd, int cmd, long arg);
```

```
int fcntl(int fd, int cmd, struct flock *lock);
```

(返回值: 若成功则依赖于cmd, 若出错为-1)

- The operation is determined by "cmd".



## *fcntl* Function (cont'd)

---

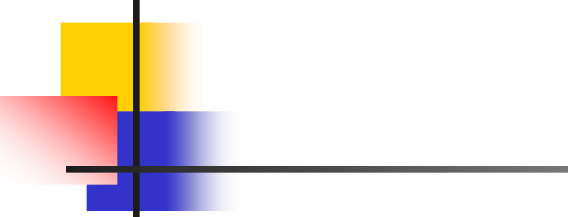
- The value of “cmd”
  - **F\_DUPFD**: Duplicate a file descriptor
  - **F\_GETFD/F\_SETFD**: Get/set the file descriptor's close-on-exec flag.
  - **F\_GETFL/F\_SETFL**: Get/set the file descriptor's flags
  - **F\_GETOWN/F\_SETOWN**: Manage I/O availability signals
  - **F\_GETLK/F\_SETLK/F\_SETLKW**: Get/set the file lock
- Example
  - dup/dup2 and fcntl



## *fcntl* Function (cont'd)

---

- The value of "cmd"
  - **F\_DUPFD**: Duplicate a file descriptor
  - **F\_GETFD/F\_SETFD**: Get/set the file descriptor's close-on-exec flag.
  - **F\_GETFL/F\_SETFL**: Get/set the file descriptor's flags
  - **F\_GETOWN/F\_SETOWN**: Manage I/O availability signals
  - **F\_GETLK/F\_SETLK/F\_SETLKW**: Get/set the file lock
- Example
  - dup/dup2 and fcntl



```
//file:fcntl
int main()
{
 pid_t pid;
 fd = open("test.txt",O_RDWR|O_APPEND);
 if (fd == -1)
 ##printf("open err/n");
 printf("fd = %d",fd);
 printf("fork!/n");
 fcntl(fd, F_SETFD, 1);
 char *s="oooooooooooooooooooo";
 pid = fork();
 if(pid == 0)
 execl("ass", "./ass", &fd, NULL);
 wait(NULL);
 write(fd,s,strlen(s));
 close(fd);
 return 0;
}
```



```
//ass 源代码
```

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

```
{
```

```
 int fd;
```

```
 printf("argc = %d ",argc);
```

```
 fd = *argv[1];
```

```
 printf("fd = %d",fd);
```

```
 char *s = "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz";
```

```
 write(fd, (void *)s, strlen(s));
```

```
 close(fd);
```

```
 return 0;
```

```
}
```



# *ioctl* Function

---

- Control devices

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

```
int ioctl(int d, int request, ...);
```





# Standard I/O Library

---

- File stream
- Standard I/O functions



# File Stream

---

- Stream and “FILE” structure
  - FILE\* fp;
  - Predefined pointer: stdin, stdout, stderr
- Buffered I/O
  - Three types of buffers
    - Full buffer
    - Line buffer
    - No buffer
  - setbuf/setvbuf functions



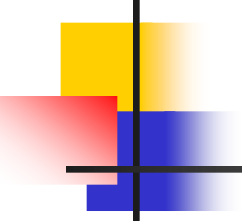
# Stream Buffering Operations

---

- Three types of buffering
  - block buffered (fully buffered)
  - line buffered
  - unbuffered
- setbuf, setvbuf functions

```
#include <stdio.h>

void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int mode, size_t
size);
```

- 
- 
- `void setbuf(FILE *stream, char *buf);`
  - `int setvbuf(FILE *stream, char *buf, int type, unsigned size);`
    - type: `_IOFBF`(满缓冲) `_IOLBF`(行缓冲)  
`_IONBF`(无缓冲)



# Standard I/O Functions

---

- Stream open/close
- Stream read/write
  - 每次一个字符的I/O
  - 每次一行的I/O
  - 直接I/O(二进制I/O)
  - 格式化I/O
- Stream reposition
- Stream flush



# Stream open/close

---

- Open a stream

```
#include <stdio.h>
```

```
FILE *fopen(const char *filename, const char *mode);
```

```
int fclose(FILE *stream);
```

- Parameter "mode"

"r": Open text file for reading.

"w": Truncate file to zero length or create text file for writing.

"a": Open for appending.

"r+": Open for reading and writing.

"w+": Open for reading and writing. The file is created if it does not exist, otherwise it is truncated.

"a+": Open for reading and appending. The file is created if does not exist.



# Stream open/close (cont'd)

---

- Close a stream

```
#include <stdio.h>
```

```
int fclose(FILE *fp);
```

(Return: 0 if success; -1 if failure)



# Input of a character

---

- `getc`, `fgetc`, `getchar` functions

`#include <stdio.h>`

`int getc(FILE *fp);`

`int fgetc(FILE *fp);`

`int getchar(void);`

(Result: Reads the next character from a stream and returns it as an **unsigned char cast to an int**, or **EOF** on end of file or error.)

- Three functions:
  - `ferror`, `feof`, `clearerr`
- `ungetc` function: push a character back to a stream.





# Output of a Character

---

- **putc, fputc, putchar functions**

```
#include <stdio.h>
```

```
int putc(int c, FILE *fp);
```

```
int fputc(int c, FILE *fp);
```

```
int putchar(int c);
```

(Return: the character if success; -1 if failure)



# Input of a Line of String

---

- fgets, gets functions

```
#include <stdio.h>
```

```
char *fgets(char *s, int size, FILE *stream);
```

```
char *gets(char *s); //not recommended.
```

- fgets: reads in at most **size-1** characters from **stream** and stores them into the buffer pointed by **s**. **Reading stops after an EOF or a new line.** A **'\0'** character is stored at the end of the buffer.



# Output of a Line of String

---

- fputs, puts functions

```
#include <stdio.h>
```

```
int fputs(const char *s, FILE *stream);
```

```
int puts(const char *s);
```



# Question: I/O Efficiency

---

- Rewrite `mycat.c`
  - read/write version
  - `getc/putc` version
  - `fgetc/fputc` version
  - `fgets/fputs` version



# mycat.c

```
#include "ourhdr.h"

#define BUFSIZE 8192

int
main(void)
{
 int n;
 char buf[BUFSIZE];

 while ((n = read(STDIN_FILENO, buf, BUFSIZE)) > 0)
 if (write(STDOUT_FILENO, buf, n) != n)
 err_sys("write error");

 if (n < 0)
 err_sys("read error");

 exit(0);
}
```

| BUFFSIZE | 用 户 CPU<br>(秒) | 系 统 CPU<br>(秒) | 时 钟 时 间<br>(秒) | 循 环 次 数   |
|----------|----------------|----------------|----------------|-----------|
| 1        | 23.8           | 397.9          | 423.4          | 1 468 802 |
| 2        | 12.3           | 202.0          | 215.2          | 734 401   |
| 4        | 6.1            | 100.6          | 107.2          | 367 201   |
| 8        | 3.0            | 50.7           | 54.0           | 183 601   |
| 16       | 1.5            | 25.3           | 27.0           | 91 801    |
| 32       | 0.7            | 12.8           | 13.7           | 45 901    |
| 64       | 0.3            | 6.6            | 7.0            | 22 951    |
| 128      | 0.2            | 3.3            | 3.6            | 11 476    |
| 256      | 0.1            | 1.8            | 1.9            | 5 738     |
| 512      | 0.0            | 1.0            | 1.1            | 2 869     |
| 1 024    | 0.0            | 0.6            | 0.6            | 1 435     |
| 2 048    | 0.0            | 0.4            | 0.4            | 718       |
| 4 096    | 0.0            | 0.4            | 0.4            | 359       |
| 8 192    | 0.0            | 0.3            | 0.3            | 180       |
| 16 384   | 0.0            | 0.3            | 0.3            | 90        |
| 32 768   | 0.0            | 0.3            | 0.3            | 45        |
| 65 536   | 0.0            | 0.3            | 0.3            | 23        |
| 131 072  | 0.0            | 0.3            | 0.3            | 12        |

```
#include "ourhdr.h"

int
main(void)
{
 int c;

 while ((c = getc(stdin)) != EOF)
 if (putc(c, stdout) == EOF)
 err_sys("output error");

 if (ferror(stdin))
 err_sys("input error");

 exit(0);
}
```

```
#include "ourhdr.h"

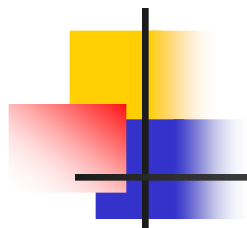
int
main(void)
{
 char buf[MAXLINE];

 while (fgets(buf, MAXLINE, stdin) != NULL)
 if (fputs(buf, stdout) == EOF)
 err_sys("output error");

 if (ferror(stdin))
 err_sys("input error");

 exit(0);
}
```





| 函 数                       | 用户CPU (秒) | 系统CPU (秒) | 时钟时间 (秒) | 程序正文字节数 |
|---------------------------|-----------|-----------|----------|---------|
| 表3-1中的最佳时间                | 0.0       | 0.3       | 0.3      |         |
| <code>fgets, fputs</code> | 2.2       | 0.3       | 2.6      | 184     |
| <code>getc, putc</code>   | 4.3       | 0.3       | 4.8      | 384     |
| <code>fgetc, fputc</code> | 4.6       | 0.3       | 5.0      | 152     |
| 表3-1中的单字节时间               | 23.8      | 397.9     | 423.4    |         |



# Binary Stream Input/Output

---

- fread/fwrite functions

```
#include <stdio.h>
```

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

(Return: the number of a items successfully read or written.)



# Binary stream input/output (cont'd)

---

- Application:

- Read/write a binary array:

```
float data[10];
```

```
if (fwrite(&data[2], sizeof(float), 4, fp) != 4)
 err_sys("fwrite error");
```

- Read/write a structure

```
struct {
```

```
 short count; long total; char name[NAMESIZE];
```

```
}item;
```

```
if (fwrite(&item, sizeof(item), 1, fp) != 1)
 err_sys("fwrite error");
```



# Formatted I/O

---

- scanf, fscanf, sscanf functions

`#include <stdio.h>`

`int scanf(const char *format, ...);`

`int fscanf(FILE *stream, const char *format, ...);`

`int sscanf(const char *str, const char *format, ...);`

- Use `fgets`, then parse the string.



# Formatted I/O (cont'd)

---

- printf, fprintf, sprintf functions

```
#include <stdio.h>
```

```
int printf(const char *format, ...);
```

```
int fprintf(FILE *stream, const char *format, ...);
```

```
int sprintf(char *str, const char *format, ...);
```



# Reposition a stream

---

- `fseek`, `ftell`, `rewind` functions
  - `#include <stdio.h>`
  - `int fseek(FILE *stream, long int offset, int whence);`
  - `long ftell(FILE *stream);`
  - `void rewind(FILE *stream);`
- `fgetpos`, `fsetpos` functions ( Introduced in ANSI C)
  - `#include <stdio.h>`
  - `int fgetpos(FILE *fp, fpos_t *pos);`
  - `int fsetpos(FILE *fp, const fpos_t *pos);`



# Flush a stream

---

- 刷新文件流。把流里的数据立刻写入文件

```
#include <stdio.h>
```

```
int fflush(FILE *stream);
```



# Stream and File Descriptor

---

- 确定流使用的底层文件描述符

```
#include <stdio.h>
int fileno(FILE *fp);
```
- 根据已打开的文件描述符创建一个流

```
#include <stdio.h>
FILE *fdopen(int fildes, const char *mode);
```





# Temporary File

---

- Create a name for a temporary file

```
#include <stdio.h>
```

```
char *tmpnam(char *s);
```

(返回值: 指向唯一路径名的指针)

- Create a temporary file

```
#include <stdio.h>
```

```
FILE *tmpfile(void);
```

(返回值: 若成功为文件指针, 若出错为NULL)



# Advanced System Calls

---

- Handling file attributes
  - stat/fstat/lstat, ...
- Handling directory



# stat/fstat/lstat functions

---

- Get file status

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

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

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

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

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

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

(Return: 0 if success; -1 if failure)



# struct stat

---

```
struct stat {
 mode_t st_mode; /*file type & mode*/
 ino_t st_ino; /*inode number (serial number)*/
 dev_t st_rdev; /*device number (file system)*/
 nlink_t st_nlink; /*link count*/
 uid_t st_uid; /*user ID of owner*/
 gid_t st_gid; /*group ID of owner*/
 off_t st_size; /*size of file, in bytes*/
 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*/
 long st_blksize; /*Optimal block size for I/O*/
 long st_blocks; /*number 512-byte blocks allocated*/
};
```



# Test macros for file types

---

- Defined in `<sys/stat.h>`

---

| <b>Macro</b>            | <b>File type</b>       |
|-------------------------|------------------------|
| <code>S_ISREG()</code>  | regular file           |
| <code>S_ISDIR()</code>  | directory              |
| <code>S_ISCHAR()</code> | character special file |
| <code>S_ISBLK()</code>  | block special file     |
| <code>S_ISFIFO()</code> | fifo                   |
| <code>S_ISLNK()</code>  | symbolic link          |
| <code>S_ISSOCK()</code> | socket                 |

---



# File Permission - Basics

---

| Perm.      | File                                          | Directory                                                                                    |
|------------|-----------------------------------------------|----------------------------------------------------------------------------------------------|
| r          | User can read contents of file                | User can list the contents of a directory                                                    |
| w          | User can change contents of file              | User can change the contents of directory                                                    |
| x          | User can execute file as a command            | User can cd to directory and can use it in PATH                                              |
| SUID       | Program runs with effective user ID of owner  |                                                                                              |
| SGID       | Program runs with effective group ID of owner | Files created in directory inherit the same group ID as the directory                        |
| Sticky bit |                                               | Only the owner of the file and the owner of the directory may delete files in this directory |



# Deep into SUID, SGID, Sticky bit

---

- Authorization in a Linux system is based on file permissions
- An SUID or SGID bit on a program elevates your authorization level while running that program to the authorization level of the owner of that program
- Typical SUID/SGID programs are **su** and **sudo**



# File permission

---

| st_mode屏蔽      | 含义                                |
|----------------|-----------------------------------|
| S_IRUSR(00400) | Read by owner                     |
| S_IWUSR(00200) | Write by owner                    |
| S_IXUSR(00100) | Execute by owner                  |
| S_IRWXU(00700) | Read, write and execute by owner  |
| S_IRGRP(00040) | Read by group                     |
| S_IWGRP(00020) | Write by group                    |
| S_IXGRP(00010) | Execute by group                  |
| S_IRWXG(00070) | Read, write and execute by group  |
| S_IROTH(00004) | Read by others                    |
| S_IWOTH(00002) | Write by others                   |
| S_IXOTH(00001) | Execute by others                 |
| S_IRWXO(00007) | Read, write and execute by others |





# File permission (cont'd)

---

| <b>st_mode</b> 屏蔽 | 含义                          |
|-------------------|-----------------------------|
| S_ISUID(04000)    | Set user ID on execution    |
| S_ISGID(02000)    | Set group ID on execution   |
| S_ISVTX(01000)    | Saved-text bit (sticky bit) |

- Example: testing file permission

```
if (buf.st_mode & S_IRUSR)
 printf(“readable by owner”);
else
 printf(“unreadable by owner”);
```



# access function

---

- 按实际用户ID和实际组ID测试文件存取权限

```
#include <unistd.h>
int access(const char *pathname, int mode);
(Return: 0 if success; -1 if failure)
```
- Parameter "mode"
  - R\_OK, W\_OK, X\_OK, F\_OK



# chmod/fchmod functions

---

- Change permissions of a file

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

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

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

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

```
(Return: 0 if success; -1 if failure)
```



# chown/fchown/lchown functions

---

- Change ownership of a file

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

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

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

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

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

```
(Return: 0 if success; -1 if failure)
```



# umask function

---

- 为进程设置文件存取权限屏蔽字，并返回以前的值

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

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

```
mode_t umask(mode_t mask);
```



# link/unlink functions

---

- Create a new link to (make a new name for) a file.

`#include <unistd.h>`

`int link(const char *oldpath, const char *newpath);`

(Return: 0 if success; -1 if failure)

- Delete a name and possibly the file it refers to.

`#include <unistd.h>`

`int unlink(const char *pathname);`

(Return: 0 if success; -1 if failure)



# symlink/readlink

---

- Create a symbolic link (named newpath which contains the sting "oldpath")

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

```
int symlink(const char *oldpath, const char *newpath);
```

(Return: 0 if success; -1 if failure)

- Read value of a symbolic link

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

```
int readlink(const char *path, char *buf, size_t bufsiz);
```

(Return: the count of characters placed in the buffer if success;  
-1 if failure)



# Handling directories

---

- mkdir/rmdir
- chdir/fchdir, getcwd
- Read a directory
  - opendir/closedir
  - readdir
  - telldir
  - seekdir





# mkdir/rmdir functions

---

- 创建一个空目录

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

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

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

(Return: 0 if success; -1 if failure)

- 删除一个空目录

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

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

(Return: 0 if success; -1 if failure)



# chdir/fchdir functions

---

- Change working directory

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

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

```
int fchdir(int fd);
```

(Return: 0 if success; -1 if failure)

- 当前工作目录是进程的属性，所以该函数只影响调用 **chdir** 的进程本身
  - **cd(1)** command



# getcwd function

---

- 获得当前工作目录的绝对路径

`#include <unistd.h>`

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

(返回值: 若成功则为buf, 若出错则为NULL)



# Read a directory

---

- Data structures
  - DIR, struct dirent
- Operations
  - opendir/closedir
  - readdir
  - telldir
  - seekdir



# Data structures

---

- DIR

- The data type of directory stream objects
- in `<dirent.h>`

```
typedef struct __dirstream DIR;
```

- struct dirent

- Directory item
- Defined in `<dirent.h>`

```
ino_t d_ino; /* inode number */
char d_name[NAME_MAX + 1]; /* file name */
```



# Operations

---

- 目录的打开、关闭、读、定位

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

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

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

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

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

```
off_t telldir(DIR *dir);
```

```
void seekdir(DIR *dir, off_t offset);
```



# A directory scanning program

---

```
DIR *dp;
struct dirent *entry;

if ((dp = opendir(dir)) == NULL)
 err_sys(...);
while ((entry = readdir(dp)) != NULL) {
 lstat(entry->d_name, &statbuf);
 if (S_ISDIR(statbuf.st_mode))
 ...
 else
 ...
}
closedir(dp);
```



# File lock

---

- 锁起的作用
  - 几个进程同时操作一个文件
- 锁放在哪里
  - 理论
  - 实践





# 文件锁分类

---

- 记录锁
- 劝告锁
  - 检查，加锁有应用程序自己控制
- 强制锁
  - 检查，加锁由内核控制
  - 影响[`open()` `read()` `write()`]等
- 共享锁
- 排他锁



# 特殊类型

---

- 共享模式强制锁
- 租借锁



# 标志位

---

- `mount -o mand /dev/sdb7 /mnt`
- `super_block`
  - `s_flags`
- `MS_MANDLOCK`



# fcntl记录锁

---

- 用于记录锁的**fcntl**函数原型

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

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

```
int fcntl(int fd, int cmd, struct flock *lock);
```

(返回值: 若成功则依赖于cmd, 若出错为-1)



# struct flock

---

```
struct flock{
 ...
 short l_type; /* Type of lock: F_RDLCK, F_WRLCK, F_UNLCK */
 short l_whence; /* How to interpret l_start: SEEK_SET, SEEK_CUR,
 SEEK_END */
 off_t l_start; /* Starting offset for lock */
 off_t l_len; /* Number of bytes to lock */
 pid_t l_pid; /* PID of process blocking our lock (F_GETLK only) */
 ...
}
```



## cmd参数

---

- cmd参数的取值
  - F\_GETLK: 获得文件的封锁信息
  - F\_SETLK: 对文件的某个区域封锁或解除封锁
  - F\_SETLKW: 功能同F\_SETLK, wait方式。



# 其它封锁命令

---

- lockf函数

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

```
int lockf(int fd, int cmd, off_t len);
```