第五章 标准I/O库

1. 流和FILE对象

- 标准I/O库的操作是围绕流（Stream）进行的，当用标准I/O库打开或创建一个文件时，一个流和一个文件已经相结合
- 标准I/O函数fopen返回一个FILE对象指针，该结构包含：文件描述符，指向流存储的指针，缓存长度，缓存内字符数，出错标志等。

2. 标准输入、输出、出错

- 文件描述符

  STDIN_FILENO

  STDOUT_FILENO

  STDERR_FILENO

- <stdio.h>
- 三个标准I/O流预定义文件指针：

  stdin, stdout, stderr

3. 缓存

- 全缓存
  - 当填满标准I/O缓存后才进行实际I/O操作
- 行缓存
  - 当在输入和输出中遇到新行符时，标准I/O库执行I/O操作
- 不带缓存
  - 标准I/O库不对字符进行缓存
  - 标准出错流stderr通常是不带缓存的

3. 缓存

- 更改缓存类型

```
#include <stdio.h>

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

_IOFBF	全缓存
_IOLBF	行缓存
_IONBF	不带缓存

3. 缓存

- 更改缓存类型

函　数	*mode*	*buf*	缓存及长度	缓存的类型
setbuf		nonnull	长度为BUFSIZ的用户缓存	全缓存或行缓存
		NULL	(无缓存)	不带缓存
setvbuf	_IOFBF	nonnull	长度为*size*的用户缓存	全缓存
		NULL	合适长度的系统缓存	
	_IOLBF	nonnull	长度为*size*的用户缓存	行缓存
		NULL	合适长度的系统缓存	
	_IONBF	忽略	无缓存	不带缓存

3. 缓存

- 更改缓存类型

```
#include <stdio.h>

int  fflush(FILE  *fp);
```

4. 打开流

- 打开一个标准I/O流

```
#include <stdio.h>
FILE  *fopen(const  char *pathname,  const  char *type);
FILE *freopen(const char *pathname, const char *type, FILE *fp);
FILE *fdopen(int  filedes,  const char *type);
```

type	说　明
r 或 rb	为读而打开
w 或 wb	使文件成为0长，或为写而创建
a 或 ab	添加；为在文件尾写而打开，或为写而创建
r+ 或 r+b 或 rb+	为读和写而打开
w+ 或 w+b 或 wb+	使文件为0长，或为读和写而打开
a+ 或 a+b 或 ab+	为在文件尾读和写而打开或创建

4. 打开流

- 打开一个标准I/O流

```
#include <stdio.h>
FILE  *fopen(const  char *pathname,  const  char *type);
FILE *freopen(const char *pathname, const char *type, FILE *fp);
FILE *fdopen(int  filedes,  const char *type);
```

- 当以读和写类型打开一文件时，具有限制
  - 如果中间没有fflush、fseek、fsetpos或rewind，则在输出的后面不能直接跟随输入；
  - 如果中间没有fseek、fsetpos或rewind，或一个输出操作没有达到文件尾端，则在输入后不能直接跟随输出；

4. 打开流

- 打开一个标准I/O流的六种不同方式

限　制	r	w	a	r+	w+	a+
文件必须已存在	•			•		
擦除文件以前的内容		•			•	
流可以读	•			•	•	•
流可以写		•	•	•	•	•
流只可在尾端处写			•			•

4. 打开流

- 关闭一个打开的流

```
#include <stdio.h>

int  fclose(FILE  *fp);
```

5. 读和写流

- 三种不同类型的非格式化I/O
  - 每次一个字符的I/O (getc, putc)
  - 每次一行的I/O (fgets, fputs)
  - 直接I/O (fread, fwrite)

5. 读和写流

- 每次一个字符I/O

```
#include <stdio.h>

int getc(FILE *fp);
int fgetc(FILE *fp);
int getchar(void);

int ferror(FILE *fp);
int feof(FILE *fp);

void clearerr(FILE *fp);
int ungetc(int c, FILE *fp);
```

5. 读和写流

- 每次一个字符I/O

```
#include <stdio.h>

int putc(int c, FILE *fp);
int fputc(int c, FILE *fp);
int putchar(void);
```

5. 读和写流

- 每次一行I/O

```
#include <stdio.h>

char *fgets(char *buf, int n, FILE *fp);
char *gets(char *buf);

int fputs(const char *str, FILE *fp);
int puts(const char *str);
```

6. 标准I/O的效率

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	
fgets,fputs	2.2	0.3	2.6	184
getc,putc	4.3	0.3	4.8	384
fgetc,fputc	4.6	0.3	5.0	152
表3-1中的单字节时间	23.8	397.9	423.4	

7. 二进制I/O


```
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);

struct {
    short count;
    long total;
    char name[NAMESIZE];
} item;
fwrite(&items, sizeof(item), 1, fp);
```

8. 定位流


```
#include <stdio.h>

long ftell(FILE *fp);
int fseek(FILE *fp, long offset, int whence);

void rewind(FILE *fp);

int fgetpos(FILE *fp, fpos_t *pos);
int fsetpos(FILE *fp, const fpos_t *pos);
```

9. 格式化I/O

- 格式化输出

```
#include <stdio.h>

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

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

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

9. 格式化I/O

- 格式化输出

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

int vprintf(const char *format, va_list arg);

int vfprintf(FILE *fp, const char *format, va_list arg);

int vsprintf(char *buf, const char *format, va_list arg);
```

9. 格式化I/O

- 格式化输入

```
#include <stdio.h>

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

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

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

```
#include    "ourhdr.h"

void    pr_stdio(const char *, FILE *);

int
main(void)
{
    FILE    *fp;

    fputs("enter any character\n", stdout);
    if (getchar() == EOF)
        err_sys("getchar error");
    fputs("one line to standard error\n", stderr);

    pr_stdio("stdin",  stdin);
    pr_stdio("stdout", stdout);
    pr_stdio("stderr", stderr);

    if ( (fp = fopen("/etc/motd", "r")) == NULL)
        err_sys("fopen error");
    if (getc(fp) == EOF)
        err_sys("getc error");
    pr_stdio("/etc/motd", fp);
    exit(0);
}

void
pr_stdio(const char *name, FILE *fp)
{
    printf("stream = %s, ", name);
                /* following is nonportable */
    if      (fp->_flag & _IONBF)    printf("unbuffered");
    else if (fp->_flag & _IOLBF)    printf("line buffered");
    else /* if neither of above */  printf("fully buffered");
    printf(", buffer size = %d\n", fp->_bufsiz);
}
```

10. 临时文件

- 创建临时文件

```
#include <stdio.h>

char  *tmpname(char  *ptr);

FILE  *tmpfile(void);
```

10. 临时文件

- 创建临时文件

```
main(void)
{
    char    name[L_tmpnam], line[MAXLINE];
    FILE    *fp;

    printf("%s\n", tmpnam(NULL));        /* first temp name */

    tmpnam(name);                        /* second temp name */
    printf("%s\n", name);

    if ( (fp = tmpfile()) == NULL)       /* create temp file */
        err_sys("tmpfile error");
    fputs("one line of output\n", fp);   /* write to temp file */
    rewind(fp);                          /* then read it back */
    if (fgets(line, sizeof(line), fp) == NULL)
        err_sys("fgets error");
    fputs(line, stdout);                 /* print the line we wrote */
```