

第二章：IO操作（外部文件操作）

一、标准IO与文件IO的区别

1.1 IO概念

1> IO：顾名思义就是输入输出，程序与外部设备进行信息交换的过程称为IO操作

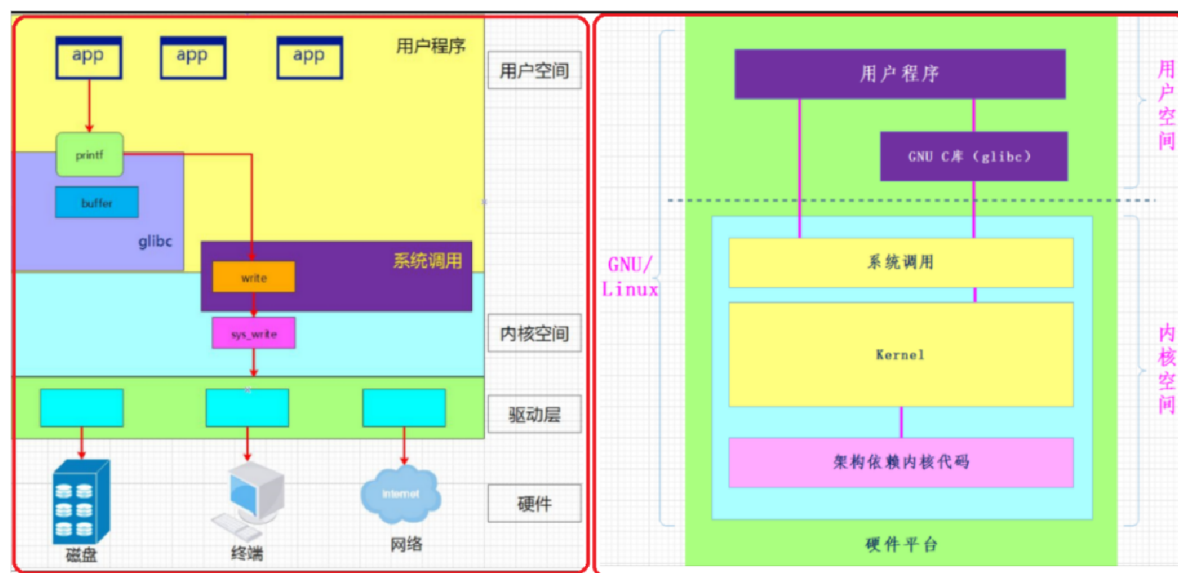
2> 最先接触的IO：`#include<stdio.h>` 标准的缓冲输入输出头文件

1.2 IO的分类

1> 标准IO：使用系统提供的库函数实现

2> 文件IO：基于系统调用完成，每进行一次系统调用，进程会从用户空间向内核空间进行一次切换，当用户空间与内核空间进行切换时，进程就会进入一次挂起状态，从而导致进程执行效率低

3> 文件IO与标准IO的区别：标准IO相比于文件IO而言，提供了缓冲区，用户可以将数据先放入缓冲区中，等到缓冲区时机到了后，统一进行一次系统调用，将数据刷入内核空间

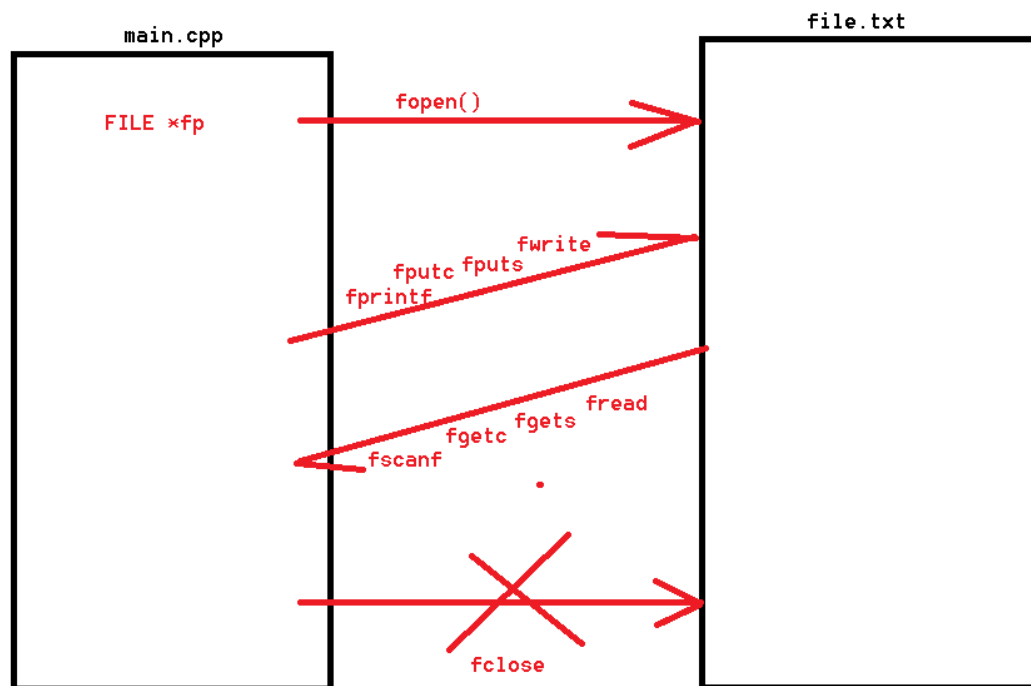


4> 标准IO接口：`printf/scanf`、`fopen/fclose`、`fgetc/fputc`、`fgets/fputs`、`fprintf/fscaf`、`fread/fwrite`、`fseek/ftell/rewind`

文件IO接口：`open/close`、`read/write`、`lseek`

二、标准IO

2.1 标准IO实现原理



2.2 FILE结构体的介绍

- 1> FILE结构体是系统提供的用于描述一个文件全部信息的结构体
- 2> FILE结构体的原型

```
struct FILE
{
    char * _IO_buf_base;           //缓冲区的起始地址
    char * _IO_buf_end;           //缓冲区终止地址

    int _fileno;                   //文件描述符，用于进行系统调用
};
```

- 3> 特殊的FILE指针，这三个指针，全部都是针对于终端文件而言的，当程序启动后，系统默认打开的三个特殊文件指针

stderr: 标准出错指针

stdin: 标准输入指针

stdout: 标准输出指针

2.3 打开文件: fopen

```
#include <stdio.h>           //该函数所在的头文件
```

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

功能：打开一个文件，并返回当前文件的文件指针

参数1：表示文件路径，是一个字符串

参数2：打开文件的方式，也是一个字符串，字符串必须以以下的字符开头

r	以只读的形式打开文件，文件光标定位在开头部分
r+	以读写的方式打开文件，文件光标定位在开头部分
w	以只写的方式打开文件，如果文件不存在，就创建文件，如果文件存在，就清空，光标定位在开头
w+	以读写的方式打开文件，如果文件不存在，就创建文件，如果文件存在，就清空，光标定位在开头
a	以追加（结尾写）的形式打开文件，如果文件不存在就创建文件，光标定位在文件末尾
a+	以读写的方式打开文件，如果文件不存在就创建文件，如果文件存在，读取操作光标定位在开头，写操作光标定位在结尾

返回值：成功返回打开的文件指针，失败返回NULL并置位错误码

```
#include "stdio.h"           //标准的输入输出头文件
```

```
int main(int argc, const char *argv[])
{
    //1、定义一个文件指针
    FILE *fp = NULL;
    //以只读的形式打开一个不存在的文件,并将返回结果存入到fp指针中
    //fp = fopen("./file.txt", "r");
    //此时会报错，原因是以只读的形式打开一个不存在的文件，是不允许的

    //以只写的方式打开一个不存在的文件,如果文件不存在就创建一个空的文件，如果文件存在就清空
    fp = fopen("./file.txt", "w");
    if(fp == NULL)
    {
        printf("fopen error\n");
        return -1;
    }
    printf("fopen success\n");

    return 0;
}
```

2.4 关闭文件：fclose

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

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

功能：关闭指定的文件

参数：要关闭的文件指针（由fopen返回的结果）

返回值：成功执行返回0，失败返回EOF并置位错误码

```
#include "stdio.h"           //标准的输入输出头文件
```

```

int main(int argc, const char *argv[])
{
    //1、定义一个文件指针
    FILE *fp = NULL;
    //以只读的形式打开一个不存在的文件,并将返回结果存入到fp指针中
    //fp = fopen("./file.txt", "r");
    //此时会报错,原因是以只读的形式打开一个不存在的文件,是不允许的

    //以只写的形式打开一个不存在的文件,如果文件不存在就创建一个空的文件,如果文件存在就清空
    fp = fopen("./file.txt", "w");
    if(fp == NULL)
    {
        printf("fopen error\n");
        return -1;
    }
    printf("fopen success\n");

    //2、关闭文件
    fclose(fp);

    return 0;
}

```

2.5 关于错误码的问题

1> 概念: 当内核提供的函数出错后, 内核空间会向用户空间反馈一个错误信息, 由于错误信息比较多也比较复杂, 系统就给每种不同的错误信息起了一个编号, 用一个整数表示, 这个整数就是错误码

2> 错误码都是大于或等于0的数字:

#define EPERM	1	/* operation not permitted */	操作受限
#define ENOENT	2	/* No such file or directory */	没有当前文件或目录
#define ESRCH	3	/* No such process */	没有进程
#define EINTR	4	/* Interrupted system call */	中断系统调用
#define EIO	5	/* I/O error */	
#define ENXIO	6	/* No such device or address */	
#define E2BIG	7	/* Argument list too long */	
#define ENOEXEC	8	/* Exec format error */	
#define EBADF	9	/* Bad file number */	
#define ECHILD	10	/* No child processes */	
#define EAGAIN	11	/* Try again */	
#define ENOMEM	12	/* Out of memory */	
#define EACCES	13	/* Permission denied */	
#define EFAULT	14	/* Bad address */	
#define ENOTBLK	15	/* Block device required */	
#define EBUSY	16	/* Device or resource busy */	
#define EEXIST	17	/* File exists */	
#define EXDEV	18	/* Cross-device link */	
#define ENODEV	19	/* No such device */	
#define ENOTDIR	20	/* Not a directory */	
#define EISDIR	21	/* Is a directory */	
#define EINVAL	22	/* Invalid argument */	
#define ENFILE	23	/* File table overflow */	
#define EMFILE	24	/* Too many open files */	
#define ENOTTY	25	/* Not a typewriter */	

```
#define ETXTBSY      26      /* Text file busy */
#define EFBIG        27      /* File too large */
#define ENOSPC       28      /* No space left on device */
#define ESPIPE       29      /* Illegal seek */
```

3> 关于错误码的处理函数 (strerror、perror)

```
#include <errno.h>    //错误码所在的头文件，里面定义了一个全局变量
int errno;

//将错误码对应的错误信息转换处理
#include <string.h>

char *strerror(int errnum);
功能：将错误码转换为错误信息描述
参数：错误码
返回值：错误信息字符串描述

//只打印错误码对应的错误信息的函数
#include <stdio.h>

void perror(const char *s);
功能：输出当前错误码对应的错误信息
参数：提示符号，会原样打印出来，并且会在提示数据后面加上冒号，并输出完后，自动换行
返回值：无
```

```
#include "stdio.h"      //标准的输入输出头文件
#include <errno.h>      //错误码所在的头文件
#include <string.h>      //字符串处理的头文件

int main(int argc, const char *argv[])
{
    //1、定义一个文件指针
    FILE *fp = NULL;

    fp = fopen("./file.txt", "r");
    if(fp == NULL)
    {
        //printf("fopen error: %d, errmsg:%s\n", errno, strerror(errno));    //
        打印错误信息，并输出错误码 2
        perror("fopen error");        //打印当前错误码对应的错误信息
        return -1;
    }
    printf("fopen success\n");

    //2、关闭文件
    fclose(fp);

    return 0;
}
```

2.6 单字符读写：fputc/fgetc

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

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

功能：从指定文件中读取一个字符数据，并以无符号整数的形式返回

参数：文件指针

返回值：成功返回读取的字符对应的无符号整数，失败返回EOF并置位错误码

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

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

功能：将指定的字符c写入到stream指向的文件中

参数1：要写入的字符对应的无符号整数

参数2：文件指针

返回值：成功返回写入字符对应的无符号整数，失败返回EOF并置位错误码

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

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

```
{
```

```
    //定义文件指针
```

```
    FILE *fp = NULL;
```

```
    if((fp = fopen("./file.txt", "w")) == NULL)    //以只写的形式打开一个文件
```

```
    {
```

```
        perror("fopen error");
```

```
        return -1;
```

```
    }
```

```
    //使用程序对文件进行读写操作（单个字符进行操作）
```

```
    //本次操作的结果是在外部文件中显示Hello，说明每写入一次，文件光标都会向后偏移
```

```
    fputc('H', fp);
```

```
    fputc('e', fp);
```

```
    fputc('l', fp);
```

```
    fputc('l', fp);
```

```
    fputc('o', fp);
```

```
    //能否从该处读取数据？不可以，为什么？因为光标此时在文件结尾处，没有数据可以被读取
```

```
    //关闭文件
```

```
    fclose(fp);
```

```
    //再次以只读的形式重新打开上一个文件
```

```
    if((fp = fopen("./file.txt", "r")) == NULL)    //以只写的形式打开一个文件
```

```
    {
```

```
        perror("fopen error");
```

```
        return -1;
```

```
    }
```

```
    char ch = 0;    //字符的搬运工，将文件中的字符，搬运到终端上
```

```
    while(1)
```

```
    {
```

```
        ch = fgetc(fp);    //从fp指向的文件中光标位置处读取一个字符，并返回
```

```
        if(ch == EOF)
```

```
        {
```

```
            break;    //说明文件已经全部被读取了，此时就可以退出循环了
```

```
        }
```

```
        printf("%c ", ch);    //在终端上打印读取的字符
```

```
    }
```

```

//关闭文件
fclose(fp);

return 0;
}

```

练习:

使用fgetc和fputc完成两个文件的拷贝工作, 实现指令cp的功能: cp srcfile destfile

```

#include<stdio.h>

int main(int argc, const char *argv[])
{
    //判断外部传参的个数是否为 3
    if(argc != 3)
    {
        printf("input file error\n");
        printf("usage:./a.out srcfile destfile\n");
        return -1;
    }

    //以只读的形式打开源文件, 以只写的形式打开目标文件
    FILE *srcfp = NULL;    //源文件文件指针
    FILE *destfp = NULL;   //目标文件文件指针
    if((srcfp = fopen(argv[1], "r")) ==NULL)
    {
        perror("srcfile open error");
        return -1;
    }
    if((destfp = fopen(argv[2], "w")) ==NULL)
    {
        perror("destfile open error");
        return -1;
    }

    //将源文件中的内容搬运到目标文件中
    char ch = 0;           //搬运工
    while(1)
    {
        ch = fgetc(srcfp);    //从源文件中读取一个字符
        if(ch == EOF)
        {
            break;            //文件全部读取结束
        }

        fputc(ch, destfp);    //将读取的字符写入到目标文件中
    }

    //关闭两个文件
    fclose(srcfp);
    fclose(destfp);

    printf("拷贝成功\n");
}

```

```
    return 0;
}
```

2.7 字符串读写：fgets/fputs

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

功能：将指定的字符串，写入到指定的文件中

参数1：要被写入的字符串

参数2：文件指针

返回值：成功返回本次写入字符的个数，失败返回EOF

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

功能：从stream指向的文件中最多读取size-1个字符到s容器中，遇到回车或文件结束，会结束一次读取，并且会将回车放入容器，最后自动加上一个字符串结束标识'\0'

参数1：字符数组容器起始地址

参数2：要读取的字符个数，最多读取size-1个字符

参数3：文件指针

返回值：成功返回容器s的起始地址，失败返回NULL

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

int main(int argc, const char *argv[])
{
    //打开文件
    FILE *fp = NULL;
    if((fp = fopen("./file.txt", "w")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    //向文件中写入多个字符
    char wbuf[128] = "";    //定义一个字符数组

    while(1)
    {
        std::cin >> wbuf;    //从终端输入一个字符串
        //将上述字符串写入文件中
        fputs(wbuf, fp);
        fputc('\n', fp);

        if(strcmp(wbuf, "quit") == 0)    //strcmp: 比较两个字符串是否一致，如果相等
            则返回0
        {
            break;
        }
    }

    //关闭文件
    fclose(fp);
}
```



```

//以只读的形式再次打开文件
if((fp = fopen("./file.txt", "r")) == NULL)
{
    perror("fopen error");
    return -1;
}

char rbuf[128] = "";
while(1)
{
    //从文件中读取数据到rbuf容器中
    char *res = fgets(rbuf, sizeof(rbuf), fp);
    if(res == NULL)
    {
        break;           //说明文件读取结束
    }

    printf("rbuf = %s\n", rbuf);    //将数据打印到终端上

}

//关闭文件
fclose(fp);

return 0;
}

```

练习:

使用fgets和fputs完成两个文件的拷贝

```

#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;

int main(int argc, const char *argv[])
{
    //判断外部是否传入两个文件
    if(argc != 3)
    {
        printf("input file error\n");
        printf("usage:./a.out srcfile destfile\n");
        return -1;
    }

    //以只读的形式打开源文件，以只写的形式打开目标文件
    FILE *srcfp = NULL;
    FILE *destfp = NULL;
    if((srcfp = fopen(argv[1], "r")) == NULL)
    {
        perror("fopen error");
        return -1;
    }
    if((destfp = fopen(argv[2], "w")) == NULL)

```

```

{
    perror("fopen error");
    return -1;
}

//定义搬运工
char buf[128] = "";
while(1)
{
    char *ptr = fgets(buf, sizeof(buf), srcfp);    //将数据从源文件中读取下来
    if(ptr == NULL)
    {
        break;    //读取结束
    }
    fputs(buf, destfp);    //将从源文件中读取的数据写入到目标文件中
}

//关闭文件
fclose(srcfp);
fclose(destfp);

printf("拷贝成功\n");

return 0;
}

```

2.8 关于标准IO的缓冲区问题

1> 缓冲区分为三种：

行缓存：和终端文件相关的缓冲区叫做行缓存，行缓冲区的大小为1024字节，对应的文件指针：stdin、stdout

全缓存：和外界文件相关的缓冲区叫做全缓存，全缓冲区的大小为4096字节，对应的文件指针：fp

不缓存：和标准出错相关的缓冲区叫不缓存，不缓存的大小为0字节，对应的文件指针：stderr

```

#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //如果缓冲区没有被使用时，求出大小为0，只有被至少使用了一次后，缓冲区的大小就被分配了
    printf("行缓存的大小为: %d\n", stdout->_IO_buf_end - stdout->_IO_buf_base);
    //行缓存的大小
    printf("行缓存的大小为: %d\n", stdout->_IO_buf_end - stdout->_IO_buf_base);
    //1024

    printf("行缓存的大小为: %d\n", stdin->_IO_buf_end - stdin->_IO_buf_base);
    //0 没有使用标准输入
    int num = 0;
    //scanf("%d", &num);    //从终端输入一个整数让如num变量中，类似于cin >> num
}

```

```

    cin >> num;                //使用了标准的输入缓冲区
    printf("行缓存的大小为: %d\n", stdin->_IO_buf_end - stdin->_IO_buf_base);
    //1024

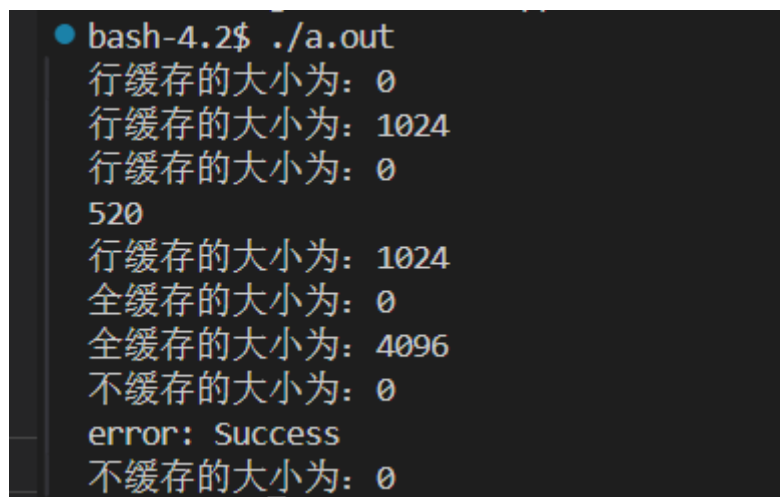
    //验证全缓存的大小为4096
    FILE *fp = NULL;
    if((fp = fopen("./aa.txt", "r")) == NULL)
    {
        perror("fopen error");
        return -1;
    }
    //未使用全缓存时, 大小为0
    printf("全缓存的大小为: %d\n", fp->_IO_buf_end - fp->_IO_buf_base);    //0
    //使用一次全缓存
    fgetc(fp);                //从文件中读取一个字符
    printf("全缓存的大小为: %d\n", fp->_IO_buf_end - fp->_IO_buf_base);    //4096
    //关闭文件
    fclose(fp);

    //验证不缓存的大小为0
    //未使用不缓存时大小为0
    printf("不缓存的大小为: %d\n", stderr->_IO_buf_end - stderr->_IO_buf_base);
    //0
    //使用一次不缓存
    perror("error");
    printf("不缓存的大小为: %d\n", stderr->_IO_buf_end - stderr->_IO_buf_base);
    //0

    return 0;
}

```

效果图:



```

● bash-4.2$ ./a.out
行缓存的大小为: 0
行缓存的大小为: 1024
行缓存的大小为: 0
520
行缓存的大小为: 1024
全缓存的大小为: 0
全缓存的大小为: 4096
不缓存的大小为: 0
error: Success
不缓存的大小为: 0

```

2> 缓冲区的刷新时机

缓冲区刷新函数: fflush

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

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

功能：刷新给定的文件指针对应的缓冲区

参数：文件指针

返回值：成功返回0，失败返回EOF并置位错误码

行缓存的刷新时机：

```
#include<iostream>
```

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

```
using namespace std;
```

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

```
{
```

```
    /*1、验证缓冲区如果没有达到刷新时机，就不会将数据进行刷新
```

```
    printf("hello world");          //在终端上打印输出一个hello world,没有到缓冲区的
```

```
    刷新时机，就不会输出数据
```

```
    while(1);          //阻塞程序不让进程结束
```

```
    */
```

```
    /*2、当程序结束后，会刷新行缓冲区
```

```
    printf("hello world");
```

```
    */
```

```
    /*3、当遇到换行时，会刷新行缓存
```

```
    printf("hello world\n");
```

```
    while(1);
```

```
    */
```

```
    /*4、当输入输出发生切换时，也会刷新行缓存
```

```
    int num = 0;
```

```
    printf("请输入>>>");          //向标准输出缓冲区中写入一组数据，没有换行符号
```

```
    scanf("%d", &num);          //cin >> num
```

```
    */
```

```
    /*5、当关闭行缓存对应的文件指针时，也会刷新行缓存
```

```
    printf("hello world");          //向标准输出缓冲区中写入数据，没有换行
```

```
    fclose(stdout);          //关闭标准输出指针
```

```
    while(1);
```

```
    */
```

```
    /*6、使用fflush函数手动刷新缓冲区时，行缓存会被刷新
```

```
    printf("hello world");
```

```
    fflush(stdout);
```

```
    while(1);
```

```
    */
```

```
    //7、当缓冲区满了后，会刷新行缓存，行缓存大小：1024字节
```

```
    for(int i=0; i<1025; i++)
```

```
    {
```

```
        printf("A");
```

```
    }
```

```
    while(1);          //防止程序结束
```

```
    return 0;
}
```

全缓存的刷新时机

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //打开一个文件
    FILE *fp = NULL;
    if((fp = fopen("./aa.txt", "r+")) == NULL)           //以读写的形式打开文件
    {
        perror("fopen error");
        return -1;
    }

    /*1、当缓冲区刷新时机未到时，不会刷新全缓存
    fputs("hello world", fp);                          //将字符串写入到文件中
    while(1);
    */

    /*2、当遇到换行时，也不会刷新全缓存
    fputs("hello world\n", fp);                        //将字符串写入到文件中
    while(1);
    */

    /*3、当程序结束后，会刷新全缓存
    fputs("hello world 你好 星球! \n", fp);
    */

    /*4、当输入输出发生切换时，会刷新全缓存
    fputs("I love China\n", fp);                      //向文件中输出一个字符串
    fgetc(fp);                                         //从文件中读取一个字符，主要是让输入输出发生切换

    while(1);
    */

    /*5、当关闭缓冲区对应的文件指针时，也会刷新全缓存
    fputs("上海欢迎你", fp);
    fclose(fp);                                       //刷新文件指针
    while(1);
    */

    /*6、当手动刷新缓冲区对应的文件指针时，也会刷新全缓存
    fputs("上海欢迎你", fp);
    fflush(fp);                                       //刷新文件指针
    while(1);
    */

    //7、当缓冲区满了后，再向缓冲区中存放数据时会刷新全缓存
    for(int i=0; i<4097; i++)
    {
        fputc('A', fp);                            //循环将单字符放入文件中
    }
}
```

```

    }
    while(1);

    return 0;
}

```

对于不缓存的刷新时机，只要放入数据，立马进行刷新

```

#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //perror("a");          //向标准出错中放入数据

    fputs("A", stderr);      //向标准出错缓冲区中写入一个字符A

    while(1);                //阻塞

    return 0;
}

```

2.9 格式化读写：fprintf/fscanf

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

功能：向指定的文件中输出一个格式串

参数1：文件指针

参数2：格式串，可以包含格式控制符，%d（整数）、%s(字符串)、%f(小数)、%lf(小数)

参数3：可变参数，输出项列表，参数个数由参数2中的格式控制符的个数决定

返回值：成功返回输出的字符个数，失败返回一个负数

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

功能：从指定的文件中以指定的格式读取数据，放入程序中

参数1：文件指针

参数2：格式串，可以包含格式控制符，%d（整数）、%s(字符串)、%f(小数)、%lf(小数)

参数3：可变参数，输入项地址列表，参数个数由参数2中的格式控制符的个数决定

返回值：成功返回读入的项数，失败返回EOF并置位错误码

```

#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //向标准输出文件指针中写入数据
    fprintf(stdout, "%d %lf %s\n", 520, 3.14, "I Love Xingqiu");

    //从标准输入中读取数据到程序中来
    int num = 0;
    fscanf(stdin, "%d", &num);          //从标准输入缓冲区中以%d的格式读取一个整数
    printf("num = %d\n", num);          //1314
}

```

```

//对外部文件进行格式化读写
FILE *fp = NULL;
if((fp = fopen("./usr.txt", "w")) == NULL)    //以只写的形式打开外部文件
{
    perror("fopen error");
    return -1;
}

//向文件中写入数据
fprintf(fp, "%s %d", "admin", 123456);    //将两个不同的数据存放到文件中

//关闭文件
fclose(fp);

//再次以只读的形式打开文件
if((fp = fopen("./usr.txt", "r")) == NULL)    //以只写的形式打开外部文件
{
    perror("fopen error");
    return -1;
}

char usrName[20] = "";    //存放用户名
int pwd = 0;    //存放密码
fscanf(fp, "%s %d", usrName, &pwd);    //从外部文件中分别读取一个字符串和一个整数放入程序中的

//usrName和pwd中

//关闭
fclose(fp);

printf("usrName = %s, pwd = %d\n", usrName, pwd);

return 0;
}

```

练习：

使用fprintf和fscanf完成注册和登录功能，要求做个小菜单，三个功能，功能1是注册，用户输入账号和密码后，将其存放到外部文件中；功能2是登录，用户输入登录账号和密码后，如果跟文件中匹配，则提示登陆成功，如果全部不匹配，则提示登录失败；功能0，表示退出。菜单可以循环调用

```

#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
using namespace std;

//定义注册函数
int do_register()
{
    //定义存储注册账户和密码的容器
    char reg_name[20] = "";
    char reg_pwd[20] = "";

    //输入注册账户和密码
    printf("请输入注册账户：");
}

```

```

fgets(reg_name, sizeof(reg_name), stdin);
reg_name[strlen(reg_name)-1] = '\0'; //将字符串最后的换行符换成结束符号
printf("请输入注册密码: ");
fgets(reg_pwd, sizeof(reg_pwd), stdin);
reg_pwd[strlen(reg_pwd)-1] = '\0';

//将账户和密码写入文件
FILE *wfp = NULL;
if((wfp = fopen("./usr.txt", "a")) == NULL) //以追加的方式打开文件
{
    perror("fopen error");
    return -1;
}
//将账户和密码写入文件
fprintf(wfp, "%s %s\n", reg_name, reg_pwd);
//关闭文件
fclose(wfp);

printf("注册成功\n");
return 0;
}

//定义登录功能
int do_login()
{
    //定义容器存储登录账户和密码
    char log_name[20] = "";
    char log_pwd[20] = "";
    char name[20] = ""; //存储从文件中读取下来的账户
    char pwd[20] = ""; //存储从文件中读取下来的密码

    //提示并输入登录账户和密码
    printf("请输入登录账户: ");
    fgets(log_name, sizeof(log_name), stdin);
    log_name[strlen(log_name)-1] = '\0'; //将字符串最后的换行符换成结束符号
    printf("请输入登录密码: ");
    fgets(log_pwd, sizeof(log_pwd), stdin);
    log_pwd[strlen(log_pwd)-1] = '\0';

    //打开文件
    FILE *rfp = NULL;
    if((rfp = fopen("./usr.txt", "r")) == NULL)
    {
        perror("fopen error");
        return -1;
    }
    //将登录账户和密码跟文件中所有账户和密码进行匹配
    while(1)
    {
        //将每一行的账户和密码读取出来
        int res = fscanf(rfp, "%s %s", name, pwd);
        if(res == EOF) //说明全部都试一遍了，他也没有成功
        {
            break;
        }

        //将读取出来的账户和密码跟登录账户和密码进行匹配
        if(strcmp(name, log_name)==0 && strcmp(pwd, log_pwd)==0)
    }
}

```



```

    {
        printf("登录成功\n");
        //关闭文件
        fclose(rfp);
        return 1;
    }

}

//关闭文件
fclose(rfp);

printf("登录失败，请重新登录\n");

return 0;
}

/*****主程序*****/
int main(int argc, const char *argv[])
{
    int menu = 0;    //存储菜单选项变量

    while(1)
    {
        system("clear");    //创建子进程调用终端指令
        printf("\t\t\t====XXX 登录界面=====\n");
        printf("\t\t\t====1、注册=====\n");
        printf("\t\t\t====2、登录=====\n");
        printf("\t\t\t====0、退出=====\n");

        printf("请输入功能选项: ");
        scanf("%d", &menu);
        getchar();    //吸收一下scanf留下的回车

        switch(menu)
        {
            case 1:
            {
                //注册
                do_register();
            }
            break;

            case 2:
            {
                //登录
                int res = do_login();
                if(res == 1)
                {
                    printf("登录成功后的界面\n");
                    //此处省略一万行程序代码
                }
            }
            break;
            case 0: exit(EXIT_SUCCESS);    //退出进程
        }
    }
}

```

```

        default:printf("您输入的功能有误，请重新输入!!!!\n");
    }
    printf("请输入任意键按回车清屏\n");
    while(getchar() != '\n');
}

return 0;
}

```

2.10 格式串转字符串存入字符数组中：sprintf/snprintf

- 1> 目前所学的函数中，printf是向终端打印一个格式串，fprintf是向外部文件中打印一个格式串
- 2> 有时候，想要将多个不同数据类型的数据，组成一个字符串放入字符数组中，此时我们就可以使用sprintf或snprintf

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

功能：将指定的格式串转换为字符串，放入字符数组中

参数1：字符数组的起始地址

参数2：格式串，可以包含多个格式控制符

参数3：可变参数

返回值：成功返回转换的字符个数，失败返回EOF

对于上述函数而言，用一个小的容器去存储一个大的转换后的字符时，会出现指针越界的段错误，为了安全起见，引入了snprintf

```
int snprintf(char *str, size_t size, const char *format, ...);
```

功能：将格式串中最多size-1个字符转换为字符串，存放到字符数组中

参数1：字符数组的起始地址

参数2：要转换的字符个数，最多为size-1

参数3：格式串，可以包含多个格式控制符

参数4：可变参数

返回值：如果转换的字符小于size时，返回值就是成功转换字符的个数，如果大于size则只转换size-1个字符，返回值就是size，失败返回EOF

```

#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    char buf[10] = "";           //定义一个字符串容器
    //需求：将多种数据类型的数据连接成一个字符串，放入字符数组中
    //sprintf(buf, "%d %s %lf", 1001, "zhangsanfeng", 99.5);

    //安全起见，我们使用snprintf
    snprintf(buf, sizeof(buf), "%d %s %lf", 1001, "zhangsanfeng", 99.5);

    //输出转换后的字符串
    printf("buf = %s\n", buf);    //1001 zhangsanfeng 99.500000

    return 0;
}

```

2.11 模块化读写 (fread/fwrite)

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

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

功能：从stream指向的文件中，读取nmemb项数据，每一项的大小为size，将整个结果放入ptr指向的容器中

参数1：容器指针，是一个void*类型，表示可以存储任意类型的数据

参数2：要读取数据每一项的大小

参数3：要读取数据的项数

参数4：文件指针

返回值：成功返回nmemb，就是成功读取的项数，失败返回小于项数的值，或者是0

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

功能：向stream指向的文件中写入nmemb项数据，每一项的大小为size，数据的起始地址为ptr

参数1：要写入数据的起始地址

参数2：每一项的大小

参数3：要写入的总项数

参数4：文件指针

返回值：成功返回写入的项数，失败返回小于项数的值，或者是0

1> 字符串的读写

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //定义文件指针，以只写的形式打开文件
    FILE *fp = NULL;
    if((fp = fopen("./test.txt", "w")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    //定义要存储的字符串
    char wbuf[128] = "";
    while(1)
    {
        //提示从终端上输入字符串
        printf("请输入>>>");
        fgets(wbuf, sizeof(wbuf), stdin); //从标准输入中读取数据
        //wbuf[strlen(wbuf)-1] = 0;        //将换行换成字符串结束标志

        //判断输入的是否为 quit
        if(strcmp(wbuf, "quit\n") == 0)
        {
            break;
        }

        //将字符串写入文件
```

```

        fwrite(wbuf, strlen(wbuf), 1, fp);    //将字符串以每个字符串为单位写入文件，
        每次写一项
        //fwrite("\n", strlen("\n"), 1, fp);    //每输入一个字符串加上换行
        //刷新缓冲区
        fflush(fp);
        printf("录入成功\n");
    }
    //关闭文件
    fclose(fp);

    //再次以只读的形式打开文件
    if((fp = fopen("./test.txt", "r")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    char rbuf[10] = "";
    int res = fread(rbuf, 1, sizeof(rbuf), fp);    //从文件中读取一个字符串
    fwrite(rbuf, 1, res, stdout);    //向标准输出中写入数据

    //关闭文件
    fclose(fp);

    return 0;
}

```

2> 整数的读写

```

#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //定义文件指针，以只写的形式打开文件
    FILE *fp = NULL;
    if((fp = fopen("./test.txt", "w")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    //定义要写入文件中的数据
    int num = 16;
    //将数据写入文件中
    fwrite(&num, sizeof(num), 1, fp);    //将num中的数据写入fp指向的文件中

    //关闭文件
    fclose(fp);

    //再次以只读的形式打开文件
    if((fp = fopen("./test.txt", "r")) == NULL)
    {

```

```

        perror("fopen error");
        return -1;
    }

    //定义接收数据的变量
    int key = 0;
    //从文件中读取数据
    fread(&key, sizeof(key), 1, fp);

    //关闭文件
    fclose(fp);

    printf("key = %d\n", key);

    return 0;
}

```

3> 结构体数据的读写

```

#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;

//定义一个学生信息
class Stu
{
public:
    char name[20];        //姓名
    int age;               //年龄
    double score;         //成绩
};

int main(int argc, const char *argv[])
{
    //定义文件指针，以只写的形式打开文件
    FILE *fp = NULL;
    if((fp = fopen("./test.txt", "w")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    //定义三个学生
    Stu s[3] = {{ "张三", 18, 98}, \
                {"李四", 20, 88}, \
                {"王五", 16, 95}};
    //将三个学生信息写入文件中
    fwrite(s, sizeof(Stu), 3, fp);

    //关闭文件
    fclose(fp);

    //再次以只读的形式打开文件
    if((fp = fopen("./test.txt", "r")) == NULL)

```

```

{
    perror("fopen error");
    return -1;
}

//定义一个对象，接收读取的结果
Stu temp;

//从文件中读取一个学生的信息
fread(&temp, sizeof(Stu), 1, fp);

//将读取的数据展示出来
printf("name:%s, age:%d, score:%.2lf\n", temp.name, temp.age, temp.score);

//关闭文件
fclose(fp);

return 0;
}

```

2.12 关于文件光标：fseek/ftell/rewind

```
int fseek(FILE *stream, long offset, int whence);
```

功能：移动文件光标位置，将光标从指定位置处进行前后偏移

参数1：文件指针

参数2：偏移量

>0:表示从指定位置向后偏移n个字节

<0:表示从指定位置向前偏移n个字节

=0:在指定位置处不偏移

参数3：偏移的起始位置

SEEK_SET: 文件起始位置

SEEK_CUR: 文件指针当前位置

SEEK_END: 文件结束位置

返回值：成功返回0，失败返回-1并置位错误码

```
long ftell(FILE *stream);
```

功能：获取文件指针当前的偏移量

参数：文件指针

返回值：成功返回文件指针所在的位置，失败返回-1并置位错误码

eg: fseek(fp, 0, SEEK_END); //将光标定位在结尾

ftell(fp); //该函数的返回值，就是文件大小

```
void rewind(FILE *stream);
```

功能：将文件光标定位在开头：fseek(fp, 0, SEEK_SET);

参数：文件指针

返回值：无

```

#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;

//定义一个学生信息
class Stu

```

```

{
public:
    char name[20];          //姓名
    int age;                //年龄
    double score;           //成绩
};

int main(int argc, const char *argv[])
{
    //定义文件指针，以只写的形式打开文件
    FILE *fp = NULL;
    if((fp = fopen("./test.txt", "w+")) == NULL)
    {
        perror("fopen error");
        return -1;
    }

    //定义三个学生
    Stu s[3] = {"张三", 18, 98}, \
               {"李四", 20, 88}, \
               {"王五", 16, 95}};

    //将三个学生信息写入文件中
    fwrite(s, sizeof(Stu), 3, fp);

    //求出文件的大小
    printf("此时文件的大小为: %ld\n", ftell(fp));          //文件大小

    //将光标移动到开头位置
    //fseek(fp, 0, SEEK_SET);

    //将光标直接定位到第二个学生信息前，但是此时光标在最后
    //fseek(fp, sizeof(Stu), SEEK_SET);          //将光标从开头后移一个学生空间的内容
    fseek(fp, -sizeof(Stu)*2, SEEK_CUR);          //将光标从当前位置向前偏移两个学生空间的内容

    //定义一个对象，接收读取的结果
    Stu temp;

    //从文件中读取一个学生的信息
    fread(&temp, sizeof(Stu), 1, fp);

    //将读取的数据展示出来
    printf("name:%s, age:%d, score: %.2lf\n", temp.name, temp.age, temp.score);

    //关闭文件
    fclose(fp);

    return 0;
}

```

补充知识：

1> 关于追代码工具ctags的使用

- 1、安装ctags: `sudo yum install ctags`
- 2、进入 `/usr/include` 目录下, 执行 `sudo ctags -R` 指令
- 3、该目录下会多出一个tags的索引文件
- 4、追相关内容的指令: `vi -t 内容`
继续向后追内容: `ctrl +]`
退出追代码工具: 底行模式下输入q, 回车

2> vscode中关于自定义片段的设置

- 1、在vscode中使用组合键: `ctrl + shift + p`调出控制面板
- 2、输入 `user snippets` 回车
- 3、输入自定义片段文件名称, 打开自定义文件
- 4、输入以下程序

```
"Simple C++ Program": {
  "prefix": "main",
  "body": [
    "#include<iostream>",
    "using namespace std;",
    "int main(int argc, const char *argv[])",
    "{",
    "    $1",
    "    return 0;",
    "}"
  ],
  "description": "Simple C++ program template"
}
```

3> 补充一个头文件中 可以包含多个头文件内容

- 1、创建一个头文件, 例如 `myhead.h`
- 2、将能用到的所有头文件都放入该文件中
- 3、将`myhead.h`文件, 移动到根目录下的usr目录中的include目录下
`sudo mv myhead.h /usr/include`
- 4、常用的头文件

```
#include<iostream>           //C++标准输入输出流
#include<string>              //C++字符串类对象
#include<iomanip>             //C++中的格式化输入输出
#include<stdio.h>             //C语言的输入输出头文件
#include<math.h>              //C语言的数据函数头文件
#include<string.h>            //C语言字符串处理头文件
#include<stdlib.h>            //C语言库头文件
```
- 5、重新设置用户的自定义片段

```
"Simple C++ Program": {
  "prefix": "main",
  "body": [
    "#include<myhead.h>",

    "int main(int argc, const char *argv[])",
    "{",
    "    $1",
    "    return 0;",
    "}"
  ],
}
```



```
"description": "Simple C++ program template"
}
```

三、文件IO

就是通过系统调用（内核提供的函数）实现，只要使用的文件IO接口，那么进程就会从用户空间向内核空间进行一次切换。标准IO的实现中，也是在内部调用了文件IO操作。该操作效率较低，因为没有缓冲区的概念。文件IO常用的操作：**open**、**close**、**read**、**write**、**lseek**

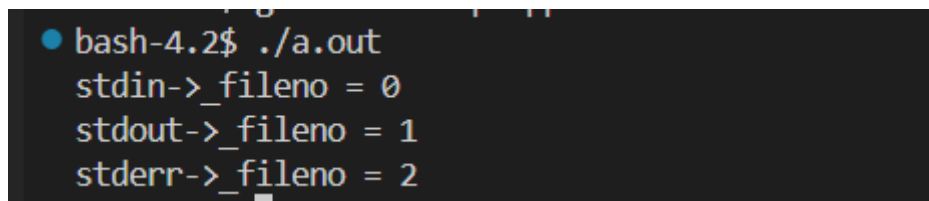
3.1 文件描述符

- 1> 文件描述符的本质是一个大于等于0的整数，在使用open函数打开文件时，就会产生一个用于操作文件的句柄，这就是文件描述符
- 2> 在一个进程中，能够打开的文件描述符是有限制的，一般是1024个，[0,1023],可以通过指令 `ulimit -a`进行查看，如果要更改这个限制，可以通过指令 `ulimit -n 数字`，进行更改
- 3> 文件描述符的使用原则一般是最小未分配原则
- 4> 特殊的文件描述符：0、1、2，这三个文件描述符在一个进程启动时就默认被打开了，分别表示标准输入、标准输出、标准错误

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc, const char *argv[])
{
    //分别输出标准输入、标准输出、标准出错文件指针对应的文件描述符
    printf("stdin->_fileno = %d\n", stdin->_fileno);           //0
    printf("stdout->_fileno = %d\n", stdout->_fileno);         //1
    printf("stderr->_fileno = %d\n", stderr->_fileno);         //2

    return 0;
}
```

效果图



```
bash-4.2$ ./a.out
stdin->_fileno = 0
stdout->_fileno = 1
stderr->_fileno = 2
```

3.2 打开文件：open

```
#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);
```

功能：打开或创建一个文件，并返回该文件的文件描述符，返回文件描述符的规则是最小未分配原则

参数1：要打开的文件文件路径

参数2：打开模式

以下三种方式必须选其一：**O_RDONLY**(只读)、**O_WRONLY**（只写）、**O_RDWR**（读写）

以下的打开方式可以有零个或多个，跟上述的方式一起用位或运算连接到一起

O_CREAT:用于创建文件，如果文件不存在，则创建文件，如果文件存在，则打开文件，如果flag中包含了该模式，则函数的第三个参数必须要加上

O_APPEND:以追加的形式打开文件，光标定位在结尾

O_TRUNC:清空文件

O_EXCL:常跟**O_CREAT**一起使用，确保本次要创建一个新文件，如果文件已经存在，则open

函数报错

eg:

"w": **O_WRONLY** | **O_CREAT** | **O_TRUNC**

"w+": **O_RDWR** | **O_CREAT** | **O_TRUNC**

"r": **O_RDONLY**

"r+": **O_RDWR**

"a": **O_WRONLY** | **O_CREAT** | **O_APPEND**

"a+": **O_RDWR** | **O_CREAT** | **O_APPEND**

参数3：如果参数2中的flag中有**O_CREAT**时，表示创建新文件，参数3就必须给定，表示新创建的文件的权限

如果当前参数给定了创建的文件权限，最终的结果也不一定是参数3的值，系统会用你给定的参数3的值，与系统的umask取反的值进行位与运算后，才是最终创建文件的权限（**mode & ~umask**）

当前终端的umask的值，可以通过指令 **umask** 来查看，一般默认为 **0022**，表示当前进程所在的组中对该文件没有写权限，其他用户对该文件也没有写权限

当前终端的umask的值是可以更改的，通过指令：**umask** 数字进行更改，这种方式只对当前终端有效

普通文件的权限一般为：**0644**，表示当前用户没有可执行权限，当前组中其他用户和其他组中的用户都只有读权限

目录文件的权限一般为：**0755**，表示当前用户具有可读、可写、可执行，当前组中其他用户和其他组中的用户都没有可写权限

注意：如果不给权限，那么当前创建的权限会是一个随机值

返回值：成功返回打开文件的文件描述符，失败返回-1并置位错误码

fopen() mode	open() flags
r	O_RDONLY
w	O_WRONLY O_CREAT O_TRUNC
a	O_WRONLY O_CREAT O_APPEND
r+	O_RDWR
w+	O_RDWR O_CREAT O_TRUNC
a+	O_RDWR O_CREAT O_APPEND

3.3 文件关闭: close

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

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

功能: 关闭文件描述符对应的文件

参数: 文件描述符

返回值: 成功返回0, 失败返回-1并置位错误码

```
#include<myhead.h>
```

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

```
{
```

```
    //1、定义文件描述符, 对于文件IO而言, 句柄就是文件描述符
```

```
    int fd = -1;
```

```
    //以只读的形式创建文件, 如果文件不存在则创建文件
```

```
    //如果创建文件时没有给权限, 则该文件的权限是随机权限
```

```
    //如果创建文件时, 给定了文件的权限, 则文件最终的权限是 给定的 mode&~umask
```

```
    if((fd = open("./tt.txt", O_WRONLY|O_CREAT, 0644)) == -1)
```

```
    {
```

```
        perror("open error");
```

```
        return -1;
```

```
    }
```

```
    printf("open success fd = %d\n", fd);
```

数为3

```
    //3, 由于0、1、2已经被使用, 所以该
```

```
    //关闭文件
```

```
    close(fd);          //关闭fd引用的文件
```

```
    return 0;
```

```
}
```

3.4 读写操作: read、write

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

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

功能: 从fd文件描述符引用的文件中读取count的字符放入buf对应的容器中

参数1: 已经打开文件对应的文件描述符

参数2: 容器的起始地址

参数3: 要读取的字符个数

返回值: 成功返回读取的字符个数, 这个个数可能会小于count的值, 失败返回-1并置位错误码

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

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

功能: 将buf容器中的count个数据, 写入到fd引用的文件中

参数1: 已经打开的文件的文件描述符

参数2: 要写入的数据的起始地址

参数3: 要写入数据的个数

返回值: 成功返回写入的字符个数, 这个个数可能小于count, 失败返回-1并置位错误码

```

#include<myhead.h>
int main(int argc, const char *argv[])
{
    //1、定义文件描述符，对于文件IO而言，句柄就是文件描述符
    int fd = -1;
    //以只读的形式创建文件，如果文件不存在则创建文件
    //如果创建文件时没有给权限，则该文件的权限是随机权限
    //如果创建文件时，给定了文件的权限，则文件最终的权限是 给定的 mode&~umask
    if((fd = open("./tt.txt", O_WRONLY|O_CREAT, 0644)) == -1)
    {
        perror("open error");
        return -1;
    }

    printf("open success fd = %d\n", fd);           //3,由于0、1、2已经被使用，所以该
    数为3

    //对数据进行读写操作
    char wbuf[128] = "hello world";
    //将上述字符串写入文件中
    write(fd, wbuf, strlen(wbuf));

    //关闭文件
    close(fd);           //关闭fd引用的文件

    //再次以只读的形式打开文件，此时参数2中不需要使用O_CREAT，那么第三个参数也不需要了
    if((fd = open("./tt.txt", O_RDONLY)) == -1)
    {
        perror("open error");
        return -1;
    }
    printf("open success fd = %d\n", fd);           //3

    //定义接收数据容器
    char rbuf[5] = "";
    int res = read(fd, rbuf, sizeof(rbuf));           //从文件中读取数据放入rbuf中

    write(1, rbuf, res);           //向1号文件描述符中写入数据，之前读取多少，现在写入多少

    //关闭文件
    close(fd);

    return 0;
}

```

3.5 关于光标的操作：lseek

```

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

off_t lseek(int fd, off_t offset, int whence);

```

功能：移动光标的位置，并返回光标现在所在的位置

参数1：文件描述符

参数2：偏移量

>0:表示从指定位置向后偏移n个字节

<0:表示从指定位置向前偏移n个字节

=0:在指定位置处不偏移

参数3: 偏移的起始位置

SEEK_SET: 文件起始位置

SEEK_CUR: 文件指针当前位置

SEEK_END: 文件结束位置

返回值: 光标现在所在的位置

注意: `lseek = fseek+ftell`

```
#include<myhead.h>
int main(int argc, const char *argv[])
{
    //1、定义文件描述符，对于文件IO而言，句柄就是文件描述符
    int fd = -1;
    //以读写形式创建文件，如果文件不存在则创建文件，如果文件存在则清空文件
    //如果创建文件时没有给权限，则该文件的权限是随机权限
    //如果创建文件时，给定了文件的权限，则文件最终的权限是 给定的 mode&~umask
    if((fd = open("./tt.txt", O_RDWR|O_CREAT|O_TRUNC, 0644)) == -1)
    {
        perror("open error");
        return -1;
    }

    printf("open success fd = %d\n", fd);           //3,由于0、1、2已经被使用，所以该
    数为3

    //对数据进行读写操作
    char wbuf[128] = "hello world";
    //将上述字符串写入文件中
    write(fd, wbuf, strlen(wbuf));

    //此时文件光标是在文件的末尾位置

    //需求是：读取文件中的 world
    //lseek(fd, 6, SEEK_SET);           //从文件开头位置向后偏移6个字节
    lseek(fd, -5, SEEK_END);           //从文件结束位置向前偏移5个字节

    //定义接收数据容器
    char rbuf[5] = "";
    int res = read(fd, rbuf, sizeof(rbuf));       //从文件中读取数据放入rbuf中

    write(1, rbuf, res);           //向1号文件描述符中写入数据，之前读取多少，现在写入多少

    //关闭文件
    close(fd);

    return 0;
}
```

练习：将一个bmp格式的图片信息读取出来，并将该图片的相关内容（颜色）进行更改

关于bmp图像格式可以参考：<https://upimg.baike.so.com/doc/5386615-5623077.html>

如何将windows下的文件，复制到linux系统的Centos下

1、在windows下调出cmd窗口

2、输入指令：

scp 要拷贝的文件路径 linux下的用户名@ip地址:linux下存放的地址路径

eg: scp C:\Users\鹏程万里\Desktop\wukong.bmp

zpp@192.168.31.88:/home/zpp/IO/day2

```
#include<myhead.h>
int main(int argc, const char *argv[])
{
    //以读写的形式打开文件
    int fd = -1;
    if((fd = open("./wukong.bmp", O_RDWR)) == -1)
    {
        perror("open error");
        return -1;
    }

    //获取文件的大小
    printf("文件大小为: %ld\n", lseek(fd, 0, SEEK_END));    //输出的就是文件大小

    //定义变量存储文件大小
    int pic_size = 0;
    lseek(fd, 2, SEEK_SET);    //将文件从起始位置向后偏移两个字节，跳过文件类型
    read(fd, &pic_size, 4);    //将文件头的 3--6字节的内容读取出来
    printf("pic_size = %d\n", pic_size);    //文件的大小

    //将文件光标向后偏移54字节，跳过文件头和信息头
    lseek(fd, 54, SEEK_SET);
    //定义一像素的颜色：颜色规律是蓝绿红
    unsigned char color[3] = {0, 0, 255};    //定义一个绿色
    for(int i=0; i<100; i++)    //以像素为单位遍历行数,遍历前100行
    {
        for(int j=0; j<684; j++)    //遍历所有列数
        {
            //此时的 (i,j) 定位的就是一个像素点，是一个三字节为单位的颜色点
            write(fd, color, sizeof(color));    //将当前光标所在位置的像素点变成
            绿色
        }
    }

    //关闭文件
    close(fd);

    return 0;
}
```

课后练习：

使用文件IO来完成两个文件的拷贝，可以完成两个图像的拷贝工作

