文件IO

一. 标准IO

流：stream，是标准IO的操作对象，是描述文件信息大的结构体

1. 文件读写

1）文件打开：fopen

```
#include <stdio.h>

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

//参数
1）path：文件名（绝对路径或相对路径）
2）mode：文件打开的方式
r：只读，文件必须存在
r+：读写，文件必须存在
w：只写，如果文件不存在，则创建并打开；存在则清空文件并打开
w+：读写，如果文件不存在，则创建并打开；存在则清空文件并打开
a：追加写，如果文件不存在，则创建并打开
a+:以追加读写，如果文件不存在 则创建并打开

//返回值
成功返回FILE* 类型指针(文件流指针) 失败返回NULL
```

实际操作中记得进行错误处理

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

int main()
{
    FILE *fp = NULL;
    fp = fopen("./file1/txt","r");
    if(NULL == fp)
    {
        printf("文件打开失败");
        exit(-1);
    }
    return 0;
}
```

2) 错误提示: perror

```
#include <stdio.h>

void perror(const char *s);

//参数：用户自己输入的错误信息

//功能：返回上一个函数的错误信息
```

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

int main()
{
    FILE *fp = NULL;
    fp = fopen("./file1/txt","r");
    if(NULL == fp)
    {
        perror("fopen");
        exit(-1);
    }
    return 0;
}
```

3）标准文件

程序运行时，系统自动打开3个流

（1）标准输入：stdin --FILE * stdin

（2）标准输出：stdout ---FILE *stdout

（3）标准错误输出：stderr --FILE *stderr

4）按行读取：fgets

```
#include <stdio.h>

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

//功能
从文件中读取一行

//参数
s：读取的数据存放的位置
size：最多读多大（实际只能读到size-1个字节）
stream：要读取的文件
```

注意，读取到的字符串会在末尾自动添加 '\0'，n 个字符也包括 '\0'。也就是说，实际只读取到了 n-1 个字符，如果希望读取 100 个字符，n 的值应该为 101

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

int main()
{
    FILE *fp = NULL;
    fp = fopen("./file1/txt","r");
    if(NULL == fp)
    {
        perror("fopen");
        exit(-1);
    }

    char buf[5] = "\0";
    fgets(buf,sizeof(buf),fp);
    printf("%s\n",buf);
    //再读5-1个字节
    fgets(buf,sizeof(buf),fp);
    printf("%s\n",buf);

    fclose(fp);
    return 0;
}
```

需要重点说明的是，在读取到 n-1 个字符之前如果出现了换行，或者读到了文件末尾，则读取结束。

所以，如果我们把buf的size设置足够大，就会一行一行读取。

5）数据清零 bzero

```
#include <strings.h>

void bzero(void *s, size_t n);

//功能：将地址s的前n个字节清零
```

size_t 无符号整型 32**bit**系统上等同于unsigned int，64**bit**系统上等同于unsigned long int

例如：循环读取文件数据

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

int main()
{
    FILE *fp = NULL;
    fp = fopen("./file1/txt","r");
    if(NULL == fp)
```

```
    {
        perror("fopen");
        exit(-1);
    }

    char buf[5] = "\0";
    char *pret = NULL;
    while(1)
    {
        pret = fgets(buf, sizeof(buf),fp);
        if(NULL == pret)
        {
            break;
        }
        printf("%s",buf);
        bzero(buf, sizeof(buf));
    }
    fclose(fp);
    return 0;
}
```

上一段代码的循环部分可以简写为：

```
while( (pret = fgets(buf, sizeof(buf),fp)) != NULL )
{
    printf("%s",buf);
    bzero(buf, sizeof(buf));
}
```

6）按行写入：fputs

```
#include <stdio.h>

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

//功能
向文件中写入一行

//参数
s：存放要写入的字符串
stream：要写入的文件

//返回值
成功：返回非负数
失败：返回EOF（文件结束的标志）
```

注意要把mode改成w：

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

```
int main()
{
    FILE *fp = fopen("note1.txt","w");
    if(NULL == fp)
    {
        perror("fopen");
        exit(-1);
    }
    char buf[] = "good good study";
    int ret = fputs(buf, fp);
    if(EOF == ret)
    {
        perror("fputs");
        fclose(fp);
        exit(-1);
    }
    return 0;
}
```

7）关闭文件：fclose

```
#include <stdio.h>

int fclose(FILE *fp)

//功能：关闭文件

//返回值
成功：返回0
失败：返回EOF
```

2. 按字节读写

1）读取一个字符：fgetc

```
#include <stdio.h>

int fgetc(FILE *stream)

//功能：从文件中读取一个字符

//返回
成功：读取到的字符
失败/文件结尾：EOF
```

注意：要有r权限

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

```
int main()
{
    FILE *fp = fopen("note.txt","r");
    if(NULL == fp)
    {
        perror("fopen");
        exit(-1);
    }
    char ch = fgetc(fp);
    putchar(ch);
    return 0;
}
```

2）写入一个字符：fputc

```
#include <stdio.h>

int fputc(int c, FILE *stream);

//功能:输出一个字符到文件
//参数:
      c:要输出的字符
      stream:文件
//返回值：成功则返回写入的字符 失败EOF
```

练习1:

存在文件a.txt(有内容) 读取a.txt中内容 并且将读取的内容写入b.txt cp a.txt b.txt

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

int main(int argc, char*argv[])
{
    if(argc != 3)
    {
        printf("wrong file\n");
        exit(-1);
    }
    FILE *fp_src = fopen(argv[1],"r");
    if(NULL == fp_src)
    {
        perror("fopen source");
        exit(-1);
    }

    FILE *fp_dst = fopen(argv[2],"w");
    if(NULL == fp_dst)
    {
        perror("fopen dest");
```

```
        exit(-1);
    }

    while(1)
    {
        char ch = fgetc(fp_src);
        if(EOF == ch)
        {
            break;
        }
        fputc(ch, fp_dst);
    }

    fclose(fp_src);
    fclose(fp_dst);
}
```

2）按指定大小读fread

```
#include <stdio.h>

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

//功能：按指定大小读

//参数
ptr：读到的数据存放的位置
size：每次读的大小
nmemb：读多少次
stream：从哪读

//返回值
成功：返回读到的次数
失败：返回0
```

3）测试是否到达文件末尾：feof

```
#include <stdio.h>

int feof(FILE *stream)

//返回值
如果到达文件结尾：返回true
否则返回false
```

4）按指定大小写：fwrite

```
#include <stdio.h>

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

//参数
ptr：要写入的数据储存的位置
size：每一次写入多少
nmemb：写多少次
stream：写入的文件

//返回值
成功：返回写的次数
失败：返回0
```

5）定位文件位置指针：rewind

```
#include <stdio.h>

void rewind(FILE *stream)

//功能：将文件位置指针定位到文件的起始位置
//参数：定位stream文件的位置指针
```

6）定位文件位置指针：fseek

```
#include <stdio.h>

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

//参数
stream：文件的位置指针
offset：偏移
    1）正数：向右偏移
    2）负数：向左偏移
whence：参照
    1）SEEK_SET：文件的起始位置
    2）SEEK_END：文件的末尾
    3）SEEK_CUR：当前位置

//返回值
成功：0
```

失败：-1

ntu: ~/23031/io

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

int main()
{
    char buf[10]="\0";

    // 1234567890\nEOF

    //当前文件位置指针 指向1 起始位置1
    FILE *fp = fopen("note.txt","r");

    //定位 相对文件起始位置 向右偏移2个字节
    fseek(fp,2,SEEK_SET);

    //当前文件位置指针 指向3
    fread(buf,1,2,fp);
    printf("read:%s\n",buf);//34

    bzero(buf,sizeof(buf));

    //当前文件位置指针 指向5
    fread(buf,1,2,fp);

    printf("read:%s\n",buf);//56

    //定位 相对当前位置 向右偏移1个字节
    fseek(fp,1,SEEK_CUR);

    //当前文件位置指针 指向8
    fread(buf,1,2,fp);

    printf("read:%s\n",buf);//89

    bzero(buf,sizeof(buf));
    fseek(fp,-5,SEEK_END);//1234567890\nEOF

    //当前文件位置指针 指向7
    fread(buf,1,2,fp);

    printf("readxxx:%s\n",buf);//78
    return 0;

}
```

7）返回当前文件位置指针的位置：ftell

```
#include<stdio.h>

long ftell(FILE *stream);

//功能：返回当前文件位置指针的位置(测试当前文件位置指针相对文件开头距离)
```

```
long n = ftell(fp);
printf("%ld\n",n);
```

练习2

统计某个文件内容有多少行（'\n'）

```
/*record how many lines are there in a file */

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>

int main(int argc, char*argv[])
{
    FILE *fp = fopen("test.txt","r");
    if(NULL == fp)
    {
        perror("fopen");
        exit(-1);
    }

    char ch[20] = {"\n"};
    int count = 0;
    while((fgets(ch,sizeof(ch),fp))!= NULL)
    {
        if(ch[strlen(ch)-1] == '\n')
            count++;
        //bzero(ch,sizeof(ch));
    }

    fclose(fp);
    printf("There are %d lines\n",count);

    return 0;
}
```

二.文件IO

1. 文件打开：open

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

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

//功能：打开或创建文件
```

```
//参数：
pathname：路径名
flags：打开方式
mode：访问权限，可有可无，如果参数2是O_CREAT则参数3生效

//返回值
成功：返回文件描述符
失败：-1
```

1）flag：O_CREAT

如果文件不存在，则创建；若文件存在，则打开

```
#include <sys/types.h>
#include <sys/stats.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    int fd = open("note2.txt",O_CREAT, 0777);
    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }
}
```

2）文件描述符：fd

当前尚未分配的最小的非负整数，一般从3开始。

3）标准流的文件描述符

标准输入 stdin--键盘 0

标准输出 stdout -屏幕 1

标准错误输出 stderr -屏幕 2

4）文件的权限：mode

文件实际的权限为`mode & (~umask)`

mode:

用`ls -l`查看文件的详细信息

```
ls -l //显示文件的详细信息
```

- drwxrwxr-x 2 linux linux 4096 Jul 31 23:28 hao
- -rw-rw-r-- 1 linux linux 0 Jul 26 19:49 file.c

- 文件属性 文件所有者 文件所属组 文件大小 文件创建时间 文件名

文件属性包含10位字符
第一位代表文件类型：

- d 表示这个文件是目录
- - 表示这个文件是普通文件
- c 表示这个文件是字符设备文件
- l 表示这个文件是链接文件
- b 表示这个文件是块设备文件
- s 表示这个文件是套接字文件
- p 表示这个文件是管道文件

剩下9位代表文件权限：

- 第一个rwx 代表文件所有者对文件的访问权限 读写执行
- 第二个rwx 代表文件所属组对文件的访问权限 读写执行
- 第三个r-x 代表其他用户对文件的访问权限 读和执行

改变文件权限：chmod

```
chomod +x test.c
```

或者

- rwx rwx r-x
- 111 111 101
- 7 7 5
- chmod 777 test.c

umask掩码

用途：将生成的权限掩掉

查看掩码：`umask`，默认掩码是0 002

修改掩码：`umask 0000`，将掩码修改为000

文件实际权限：mode&(~umask)

```
文件实际权限：mode&(~umask)
按位与：都为1才为1 否则为0


             000 000 010   （掩码002）
掩码取反：    111 111 101
•            111 111 111   （文件权限）

•            --------------------------
             111 111 101  --775
```

```
注意：

•                1、文件的默认权限是没有x的，即文件的最大默认权限为666（-rw-rw-rw-）

•                2、由于进入目录和目录的x权限有关，故目录的最大默认权限为777（drwxrwxrwx）

•
我们open创建的文件 默认有执行权限 如果模拟touch  需要参数3为0666
```

练习1：实现touch

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char*argv[])
{
    if(argc != 2)
    {
        printf("%s\n",argv[0]);
        exit(-1);
    }
    int fd = open(argv[1],O_CREAT,0666);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }
}
```

5）flag：O_EXCL

测试文件是否存在，如果O_CREAT时文件已经存在，则open返回错误信息

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char*argv[])
{
    if(argc != 2)
    {
        printf("%s\n",argv[0]);
        exit(-1);
    }
    int fd = open(argv[1],O_CREAT|O_EXCL,0666);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }
}
```

如果存在，输出open: File exists

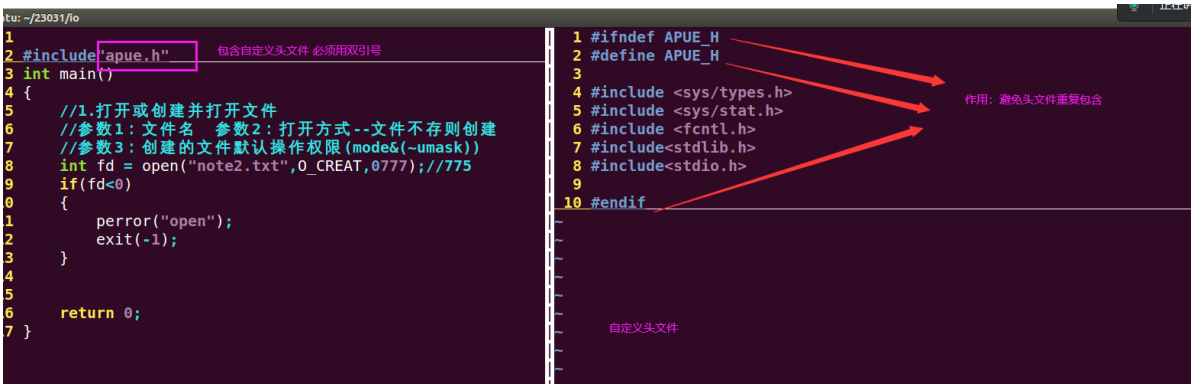
6）flag：O_TRUNC

清空文件内容

实现：如果文件不存在则创建，存在则清空打开

```
int fd = open(argv[1],O_CREAT|O_TRUNC,0666);
```

自定义头文件



2. 文件读取：read

```
#include <unistd.h>

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

//参数
fd：从fd关联的文件中读数据
buf：读到的数据有多大
count：最多读多大

//返回值
成功：返回字节数
失败：返回-1
读到结尾：返回0
```

1）flag：O_RDOLY

如果文件不存在，则创建并以只读方式打开；若文件存在，以只读方式打开

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char*argv[])
{
    if(argc != 2)
    {
        printf("%s\n",argv[0]);
        exit(-1);
    }
    int fd = open(argv[1],O_CREAT|O_RDONLY,0666);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }

    char buf[50] = "\0";

    ssize_t ret = read(fd, buf, sizeof(buf));
    printf("read: %s\n",buf);
    printf("ret: %d\n",ret);
}
```

练习2：实现cat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char*argv[])
{
    int fd = open(argv[1],O_RDONLY);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }
    char buf[50] = "\0";
    ssize_t ret;
    while(1)
    {
        ret = read(fd, buf, sizeof(buf));
        if(ret <= 0)
        {
            break;
        }
        printf("%s",buf);
        bzero(buf,sizeof(buf));
    }
}
```

3. 文件写入：write

```
#include <unistd.h>

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

//功能：将数据写入文件

//参数
fd：文件描述符
buf：要写的数据存放的位置
count：想写多少

//返回值：ssize_t有符号整型
成功：实际写入的字节数
失败：-1
```

1）flag：O_WRONLY

只写方式打开文件

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int fd = open("note2.txt",O_CREAT|O_WRONLY, 0777);
    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }

    char buf[] = "come on";
    ssize_t ret = write(fd, buf, sizeof(buf));
    if(ret <0)
    {
        perror("write");
        exit(-1);
    }
    printf("ret: %d\n",ret);
}
```

练习3：键盘输入并保存

实现从键盘输入字符串 输入'0'结束

将字符串保存到文件

//1.键盘输入字符串： scanf("",buf)

//2.将字符串写入文件：write(fd,buf)

```
int main()
{
    char *ch;
    int i = 0;
    while(ch[i-1] != '0')
    {
        scanf("%c",&ch[i]);
        getchar();
        i++;
    }
    ch[i] = '\0';
```

```
    //printf("%s\n",ch);

    int fd = open("note.txt",O_CREAT|O_WRONLY, 0777);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }

    ssize_t ret = read(fd, ch, sizeof(ch));
    if(ret < 0)
    {
        perror("read");
        exit(-1);
    }
}
```

2）flag：O_APPEND

以追加方式打开文件 读写会从文件末尾开始

```
int main()
{
    int fd = open("note.txt",O_APPEND|O_WRONLY);
    if(fd<0)
    {
        perror("open");
        exit(-1);
    }
    char buf[] = "emo";
    write(fd, buf, sizeof(buf));

}
```

4. 文件关闭：close

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

- int close(int fd);
- 功能：关闭文件fd
- 返回值：成功返回0 失败-1

5. 文件位置指针定位：lseek

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

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

//功能：文件位置指针定位

//参数：
fd：文件
offset:偏移
whence:参照
    SEEK_SET 文件起始位置   lseek(fd,2,SEEK_SET);//相对文件起始位置 向右偏移2个字节
    SEEK_END 文件的末尾    lseek(fd,-2,SEEK_END);//相对文件末尾 向左偏移2个字节
    SEEK_CUR 当前位置      lseek(fd,3,SEEK_CUR);//相对当前位置 向右偏移3个字节


//返回值：
成功返回文件位置指针相对于文件起始位置的偏移
失败-1
```

1）flag: O_RDWR

以读写方式打开

例子：把结构体数据写入文件，之后从文件中读取数据写入结构体

重点：write(fd, &x, sizeof(x));

//定位到文件起始位置
lseek(fd, 0, SEEK_SET);

```
typedef struct
{
    int id;
    int age;
}person_t;

int main()
{
    int fd = open("note2.txt",O_RDWR);
    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }

    person_t x;
    scanf("%d%d",&x.id, &x.age);
    write(fd, &x, sizeof(x));
```

```
    //定位到文件起始位置
    lseek(fd, 0, SEEK_SET);

    person_t y;
    read(fd, &y, sizeof(y));

    printf("read: %d %d\n",y.id, y.age);
}
```

练习4

键盘输入人的姓名和年龄 将信息存入文件 并读取显示

typedef struct person

{

  char name[20];

  int age;

}person_t;

person_t x;

scanf(“”,x.name,x&.age)

write(fd,&x,sizeof(x));

要求：文件名字从命令行获取

```
typedef struct
{
    char name[20];
    int age;
}person_t;

#define N 3

int main(int argc, char *argv[])
{
    if(argc != 2)
    {
        printf("not enough arguements\n");
        exit(-1);
    }
    int fd = open(argv[1],O_RDWR);
    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }

    person_t x;
```

```
    for(int i = 0; i<N; i++)
    {
        scanf("%s%d",x.name, &x.age);
        write(fd,&x, sizeof(x));
    }

    lseek(fd, 0, SEEK_SET);

    person_t y;
    for(int i = 0; i<N; i++)
    {
        read(fd,&y, sizeof(y));
        printf("%s %d\n", y.name, y.age);
    }
    close(fd);

}
```

练习5

以读写方式打开文件a.txt（已经存在 有内容） 在文件开始写入信息"hello"
在文件结尾写入"2018-4-2 16:22"， **//将文件位置指针定位到文件结尾**

再将整个文件内容读取并打印到终端 **//将文件位置指针定位到文件起始位置**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>

int main(int argc, char*argv[])
{
    int fd = open("a.txt",O_RDWR);
    if(fd <0)
    {
        perror("open");
        exit(-1);
    }
    lseek(fd, 0, SEEK_SET);

    char ch[] = "hello";
    ssize_t size = write(fd, ch, sizeof(ch));

    lseek(fd, 0, SEEK_END);
    char ch2[] = "2018-4-2 16:22";
    ssize_t size2 = write(fd, ch2, sizeof(ch2));

    lseek(fd, 0, SEEK_SET);
    char mych[50] = "\0";
    ssize_t ret;
```

```
    while(1)
    {
        ret = read(fd, mych, sizeof(mych));
        if(ret <= 0)
            break;
        printf("%s",mych);
        bzero(mych,sizeof(mych));
    }
}
```

三. 库

1. 概念

本质上来说库是一种可执行代码的二进制形式，可以被操作系统载入内存执行。

由于windows和linux的本质不同，因此二者库的二进制是不兼容的。

1）可执行的二进制代码，可被OS载入内存执行
2）库函数为了实现某个功能而封装起来的API集合//printf(“hello”)
3）提供统一的编程接口，更加便于应用程序的移植

2. 分类

1）静态库

命名规则： 静态库的名字一般为libxxxx.a，其中xxxx是该库的名称，lib开头 后缀名为.a 例如：libFun.a

制作步骤：

step1: 编写源文件

```
// add.c
#include"cal.h"
int add(int a, int b)
{
    return a+b;
}
```

```
// sub.c
#include"cal.h"
int sub(int a, int b)
{
    return a-b;
}
```

step2：做函数声明

```
// cal.h
#ifndef CAL_H
#define CAL_H

#include<stdio.h>
int add(int a, int b);
int sub(int a, int b);
#endif
```

step3：编写main函数

```
#include "cal.h"
#include <stdio.h>
int main()
{
    printf("%d\n",add(1,2));
    printf("%d\n",sub(1,2));
}
```

step4：将各个.c 分别生成.o目标文件

```
gcc -c add.c -o add.o
gcc -c  sub.c -o sub.o
```

step5：把所有的.o文件打包，打包工具是ar

```
ar crv libfun.a add.o sub.o
//输出的文件叫做libfun.a
```

step6：将main.c连接到库

```
-L 指定库所在的路径(相对main.c 路径)
-l 指定库名

gcc main.c -L . -lfun -o main.out
// .表示当前路径

./main.out
```

注意：这样之后，哪怕只有一个main.out，其他的文件都删掉，程序也会正常执行

2）动态库

共享库

命名规则：动态库的名字一般为libxxxx.so.major.minor，lib开头 后缀名为.so xxxx是该库的名称，major是主版本号， minor是副版本号

制作步骤：

step1: 编写源文件

```
// add.c
#include"cal.h"
int add(int a, int b)
{
    return a+b;
}
```

```
// sub.c
#include"cal.h"
int sub(int a, int b)
{
    return a-b;
}
```

step2：做函数声明

```
// cal.h
#ifndef CAL_H
#define CAL_H

#include<stdio.h>
int add(int a, int b);
int sub(int a, int b);
#endif
```

step3：编写main函数

```
#include "cal.h"
#include <stdio.h>
int main()
{
    printf("%d\n",add(1,2));
    printf("%d\n",sub(1,2));
}
```

step4：将各个.c 分别生成.o目标文件

```
gcc -c add.c -o add.o
gcc -c  sub.c -o sub.o
```

step5：把所有的.o文件打包

```
gcc -shared -fPIC add.o sub.o -o libmyfun.so
```

生成绿色的（可执行）的libmyfun.so

step6：将main.c连接到库

```
将main.c 连接到库
sudo cp libmyfun.so /usr/lib //指定路径也不行 必须放这里
gcc main.c -lmyfun
./a.out
```

如果删除当前目录的libmyfun.so，程序可以继续运行（因为会自动找`/usr/lib`目录。

3. 动态库和静态库区别

```
静态库优点：--以空间换时间

静态库被打包到应用程序中 加载速度快

发布程序无需提供静态库 移植方便


缺点：

浪费内存

更新、发布麻烦



动态库：--以时间换空间

优点：

 可实现进程间资源共享(共享库的好处是，不同的应用程序如果调用相同的库，那么在内存里只需要有一份该共享库的实例

)

 程序升级简单

缺点：

加载速度比静态库慢

发布程序需要提供依赖的动态库
```

四. 缓冲区

什么是缓冲区：

内存空间的一部分，相当于在内存空间预留了一些存储空间，用来缓冲输入/输出的数据，这部分预留的空间就叫做缓冲区，显然缓冲区是具有一定大小的。

当计算机的高速部件与低速部件通讯时,必须将高速部件的输出暂存到某处,以保证高速部件与低速部件相吻合。通常情况下，就是为了高效的处理我们的cpu和i/o设备之间的交互

1）用户空间的缓存：

我们的程序中的缓存，就是你想从内核读写的缓存（数组）

2）内核空间的缓存：

每打开一个文件，内核在内核空间中也会开辟一块缓存

文件IO中的写即是将**用户空间**中的缓存写到**内核空间**的缓存中。
文件IO中的读即是将**内核空间**的缓存写到**用户空间**中的缓存中。

3）库缓存：

标准IO的库函数中的缓存

缓冲文件系统（高级磁盘IO）--标准io

非缓冲文件系统（低级磁盘IO）--文件io

1. 缓冲区的类型

1.1 行缓冲

遇到回车或写满 会调用系统调用函数

```
int main()
{
    printf("hello");
    while(1);
}
```

这样写不会打印hello

改变方法：加一个\n

```
int main()
{
    printf("hello\n");
    while(1);
}
```

行缓冲遇到回车或者写满才会调用系统函数

1）强制刷新缓冲区：fflush

```
 int fflush(FILE *stream);

  功能：刷新io缓存
```

```
int main()
{
    printf("hello");
    fflush(stdout);
    while(1);
}
```

stdout, stdin, stderror

1.2 全缓冲

只有缓存写满 才调用系统调用函数

```
#include"apue.h"

int main()
{
    //打开文件
    FILE *fp = NULL;
    while(1)
    {
        fp = fopen("first.txt","a");
        fwrite("hello",5,1,fp);
        sleep(1);
    }
    return 0;
}
```

```
cat first.txt
```

会发现first.txt里没有写入

改正方法1：强刷

```
#include"apue.h"

int main()
{
    //打开文件
    FILE *fp = NULL;
    fp = fopen("first.txt","a");
    while(1)
    {
        fwrite("hello",5,1,fp);
```

```
        fflush(fp);
        sleep(1);
    }
    return 0;
}
```

改正方法2：不停开启关闭文件

```
#include"apue.h"

int main()
{
    //打开文件
    FILE *fp = NULL;
    while(1)
    {
        fp = fopen("first.txt","a");
        fwrite("hello",5,1,fp);
        fclose(fp)
        sleep(1);
    }
    return 0;
}
```

1.3 无缓冲

不对IO操作进行缓存，对流的读写可以立即操作实际文件

```
#include<stdio.h>

int main()
{
    fwrite("heihei",7,1,stderr);//标准错误输出  是没有缓存区
    while(1);
}
```

2. 文件IO和标准IO的本质区别

1）缓冲区不同：

标准IO在对文件操作时，首先操作缓冲区。等到缓冲区满足一定条件时，才会执行系统调用。

而文件IO不操作任何缓冲区，直接执行系统调用

2）系统开销

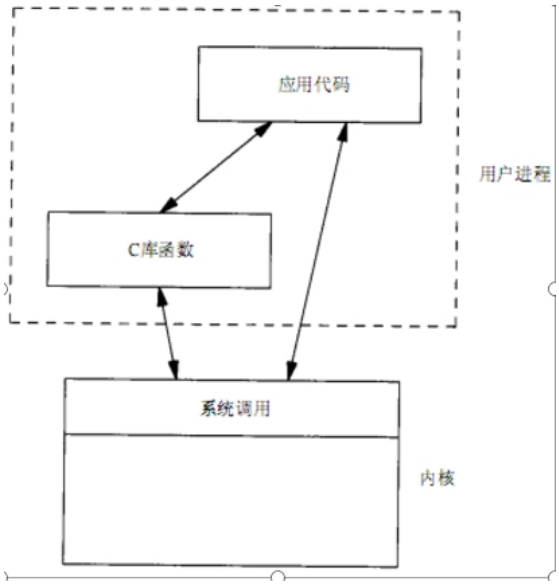
标准IO可以减少系统调用的次数，提高系统效率

（执行系统调用时，Linux必须从用户态切换到内核态，处理相应的请求，然后再返回到用户态，如果频繁地执行系统调用会增加系统的开销）

3）执行效率

采用标准I/O的函数接口 很大程度地减少了系统的调用次数，提高了执行效率

4）标准IO 一般用来操作普通文件；文件IO既可以操作普通文件也可用于多种类型的文件(管道文件)



左侧经过c库函数的是标准IO(fprintf, fwrite,..)；右侧是文件IO(read, write,...)

五. 目录IO

1. opendir

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

#include <dirent.h>

DIR *opendir(const char *name);

功能：打开目录流

参数：要打开的目录 包括路径

返回值：成功返回目录流指针 失败返回NULL
```

2. readdir

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);

//功能：读取目录

//返回值：成功struct dirent类型指针 到达结尾或失败返回NULL

struct dirent {
```

```
    ino_t      d_ino;    /* inode number */inode编号 文件编号

    off_t      d_off;    /* offset to the next dirent */

    unsigned short d_reclen;   /* length of this record */

    unsigned char  d_type;    /* type of file; not supported by all file system
types */

    char      d_name[256]; /* filename */ 文件名

};
```

3. closedir

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

#include <dirent.h>

int closedir(DIR *dirp);

功能：关闭目录
```

练习

打开当前目录并读取所有的文件名，且不显示隐藏文件(..)

```
int main()
{
    DIR *dir = opendir("./");
    struct dirent * pret = NULL;

    while(1)
    {
        pret = readdir(dir);
        if(pret == NULL)
            break;
        if(pret->d_name[0] = '.')
            continue;
        printf("%s  ",pret->d_name)
    }
    closedir(dir);
}
```

六. 文件属性和文件类型

1. 获取文件属性: stat

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

int stat(const char *path, struct stat *buf);//stat("a.txt",&s);

//功能：获取path文件属性 将其保存到buf 不能获取链接文件属性
//返回值：成功0 失败-1
```

注意，struct stat*buf是可以返回的类型

```
 struct stat {
        dev_t   st_dev;   /* ID of device containing file */
        ino_t   st_ino;   /* inode number */ inode 文件编号
        mode_t   st_mode;   /* protection */文件操作权限 文件类型
        nlink_t  st_nlink;  /* number of hard links */
        uid_t   st_uid;   /* user ID of owner */
        gid_t   st_gid;   /* group ID of owner */
        dev_t   st_rdev;   /* device ID (if special file) */
        off_t   st_size;   /* total size, in bytes */ 文件大小
        blksize_t st_blksize; /* blocksize for file system I/O */
        blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
        time_t   st_atime;  /* time of last access */
        time_t   st_mtime;  /* time of last modification */
        time_t   st_ctime;  /* time of last status change */
      };
```

练习：读取一个文件"apue.h"的属性

```
int main()
{
    struct stat s;
    int ret = stat("apue.h",&s);
    if(ret < 0)
    {
        perror("stat");
        exit(-1);
    }
    printf("%ld\n",s.st_size);
    return 0;
}
```

2. 获取文件类型mode_t st_mode

在struct stat中，有一个变量是mode_t st_mode

```
m代表st_mode

bool S_ISREG(m)  is it a regular file?  //如果该文件是常规文件 则宏返回真 否则返回假

•      S_ISDIR(m)  directory?  //目录文件

•      S_ISCHR(m)  character device? //字符设备文件

•      S_ISBLK(m)  block device?//块设备文件

•      S_ISFIFO(m) FIFO (named pipe)?//管道

•      S_ISLNK(m)  symbolic link? (Not in POSIX.1-1996.)//软链接文件

•      S_ISSOCK(m) socket? (Not in POSIX.1-1996.)//套接字文件
```

练习

获取happy文件大小并说明文件类型

```
int main()
{
    struct stat s;

    //获取文件属性
    int ret = stat("happy",&s);
    if(ret<0)
    {
        perror("stat");
        exit(-1);
    }

    //显示文件大小
```

```
    printf("%ld\n",s.st_size);

    //如果为真 说明是常规文件
    if(S_ISREG(s.st_mode))
    {
        puts("-");
    }
    //如果为真 说明是目录文件
    else if(S_ISDIR(s.st_mode))
    {
        puts("d");
    }
    return 0;
}
```