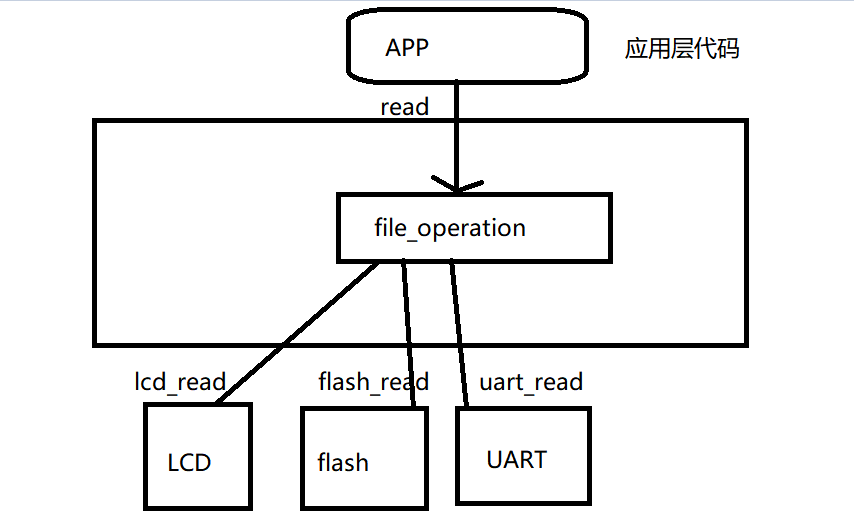
文件IO：在Linux里面一切皆文件，输入输出技术，在Linux任何设备（网络接口除外）都可以当作文件来处理



文件的7个类型：

b（block）块设备：U盘、硬盘

c（character）字符设备：串口、显示屏、触摸屏、键盘、鼠标

d（directory）目录文件

l（link）链接文件：相当于Windows里面快捷键

p（pipe）管道文件：用于进程间的通信

s（socket）套接字：网络接口，用于网络通信

-普通文件：.txt、.mp3、.mp4

C语言对文件的操作有哪些？

1、标准IO：由标准C库在系统IO的基础上进行封装来的，fread，fclose，fwrite 包含在<stdio.h>

2、系统IO：是由操作系统提供功能函数，open（打开），close（关闭），read（读取文件），write（写入文件），lseek（光标偏移），mmap（内存映射）

文件的读写效率标准IO要高于系统IO

学习一个函数：找到函数原型，去查看返回值，参数列表，结合功能说明最后再去包含头文件

1、open：打开文件

头文件：

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

函数原型：

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

int open(const char \*pathname, int flags, mode\_t mode);

参数列表：

pathname：要代开文件的路径名

flags：

O\_RDONLY：以只读的权限去打开

O\_WRONLY：以只写的权限去打开

O\_RDWR：可读可写【推荐】

--------------------------------------------------以上三个参数是互斥的

O\_APPEND：以追加的方式去打开

O\_CREAT：如果打开的文件存在，就直接打开，如果文件不存在，就会创建该文件

O\_TRUNC：打开文件之后将里面的内容清空

mode：权限0777，0666

返回值：

成功：文件描述符

失败：-1

为什么会有两个同名函数还不报错？

涉及到函数重载，在C语言里面该功能没有开放给用户使用，但是在C++里面可以使用函数重载

为什么文件描述符是从3开始的？

0：标准输入（键盘）

1：标准输出（屏幕）

2：标准出错（屏幕）

Linux系统里面文件描述符的范围0-1023，就代表最多只能打开1024个文件，在对文件操作完毕之后要进行关闭

2、close：关闭文件

头文件：

#include <unistd.h>

函数原型：

int close(int fd);

示例代码：

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

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

{

    int fd = open("../1.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0777);

    if (fd < 0)

    {

        printf("打开文件失败\n");

        return -1;

    }

    printf("fd = %d\n", fd);

    return 0;

}

3、read：读取数据

头文件：

#include <unistd.h>

函数原型：

ssize\_t read(int fd, void \*buf, size\_t count);

参数列表：

fd：文件描述符，open函数的返回值

buf：读取数据的缓冲区，就是将读取出来的内容存放在buf里面

count：要读去的字节数，一个字符占一个字节

返回值：

ssize\_t就是int类型的数据

成功：实际读取的字节数

失败：-1

注意：

返回值成功读取的字节数必须是小于等于count

在读取数据之前要先打开文件

示例代码：

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

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

{

    // 打开文件

    int fd = open("../1.txt", O\_RDWR);  // 以可读可写的方式打开

    if (fd < 0)

    {

        printf("打开文件失败\n");

        return -1;

    }

    // 读取文件内容

    char buf[128];

    ssize\_t red = read(fd, buf, 12);

    printf("buf = %s\n", buf);

    printf("red = %ld\n", red);

    return 0;

}

4、write：写入数据

头文件：

#include <unistd.h>

函数原型：

ssize\_t write(int fd, const void \*buf, size\_t count);

参数列表：

fd：文件描述符，open函数的返回值

buf：缓冲区，将buf里面的内容写入到文件fd里面去

count：要写入多少个字节

返回值：

成功：成功写入的字节数

失败：-1

示例代码：

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

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

{

    // 打开文件

    int fd = open("../1.txt", O\_RDWR | O\_TRUNC);  // 以可读可写的方式打开

    if (fd < 0)

    {

        printf("打开文件失败\n");

        return -1;

    }

    // 写入数据

    char buf[128] = "hello world";

    ssize\_t wr = write(fd, buf, strlen(buf));   // sizeof

    // 关闭文件

    close(fd);

    return 0;

}

练习1：利用write函数向屏幕写入的名字

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

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

{

    // 屏幕的文件描述符为1，不需要在去使用open打开

    write(1, "张三\n", strlen("张三\n"));

    write(STDERR\_FILENO, "李四\n", strlen("李四\n"));

read(0, buf, 10)

    return 0;

}

练习2：利用open、close、read、write函数实现文件的复制

cp 1.txt 2.txt

1、打开源文件1.txt（open）

2、打开目标文件2.txt（open）

while（1）

{

3、读取源文件1.txt内容（read）

if ()

4、写入目标文件2.txt（write）

}

5、关闭两个文件（close）

5、主函数传参：

#include <stdio.h>

// argc:表示参数的个数，注意可执行文件本身也要占据一个参数

// argv:指针数组, 代表数组的元素是指针，指向命令终端的参数，并且主函数的参数是以字符串的形式传递

int main(int argc, char const \*argv[])  // ./a.out 111 222

{

    printf("argc = %d\n", argc);    // argc = 3

    printf("argv[0] = %s\n", argv[0]);  // ./a.out

    printf("argv[1] = %s\n", argv[1]);  // 111

    printf("argv[2] = %s\n", argv[2]);  // 222

    return 0;

}

练习2讲解：

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

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

{

    if (argc != 3)

    {

        printf("请传递三个参数\n");

        return -1;

    }

    // 打开源文件, 以可读可写的权限去打开

    int src\_fd = open(argv[1], O\_RDWR);

    // 打开目标文件，以可读可写的方式打开，如果文件不存在则创建，存在则清空

    int dest\_fd = open(argv[2], O\_RDWR | O\_CREAT | O\_TRUNC, 0777);

    if (src\_fd < 0 || dest\_fd < 0)

    {

        printf("文件打开失败\n");

        return -1;

    }

    char buf[99999] = {0};

    while (1)

    {

        // 清空缓冲区

        memset(buf, 0, sizeof(buf));

        // 读取数据内容

        int red = read(src\_fd, buf, sizeof(buf));

        if(red == 0)    // 文件已经读取完毕

            break;

        // 写入到目标文件

        write(dest\_fd, buf, red);

    }

    // 关闭文件

    close(src\_fd);

    close(dest\_fd);

    return 0;

}

6、lseek：光标偏移

头文件：

#include <sys/types.h>

#include <unistd.h>

函数原型：

off\_t lseek(int fd, off\_t offset, int whence);

参数列表：

fd：文件描述符

offset：光标偏移量

whence：基准点

SEEK\_SET：文件的开头

SEEK\_CUR：代表光标当前位置

SEEK\_END：文件的末尾

返回值：

成功：新位置的偏移量

失败：-1

假设 lseek(fd, -5, SEEK\_END); 从文件末尾向前偏移五个字节

注意：如果想要光标往后偏移就将偏移量设置成正数，想要光标往前偏移就将偏移量设置成负数

练习：利用lseek函数求一个文件大小

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

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

{

    int fd = open("../002、录屏/003、shell命令.mp4", O\_RDWR);

    if (fd < 0)

    {

        printf("打开文件失败\n");

        return -1;

    }

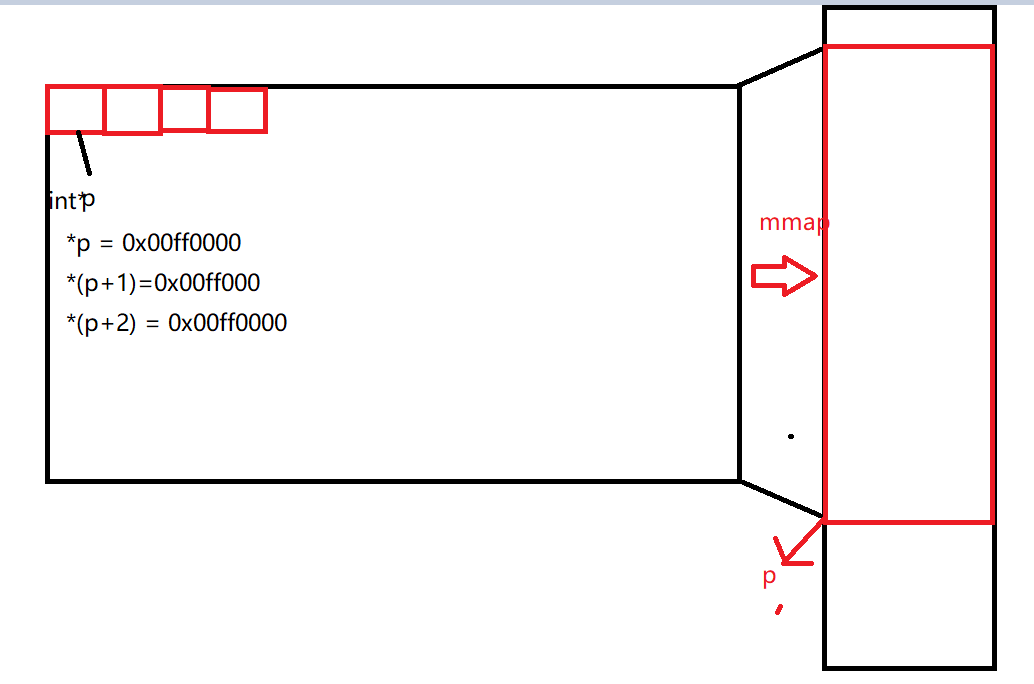
    off\_t size = lseek(fd, 0, SEEK\_END);    // off\_t = long = 2^63-1

    printf("size = %ld\n", size);

    return 0;

}

7、mmap：内存映射



头文件：

#include <sys/mman.h>

函数原型

void \*mmap(void \*addr, size\_t length, int prot, int flags,

int fd, off\_t offset); // 进行内存映射的函数

int munmap(void \*addr, size\_t length); // 解除映射

参数列表：

addr：内存映射的起始地址，一般给NULL，让系统自动去寻找

length：内存映射的大小，LCD（800\*480\*4）

prot：权限，PROT\_READ|PROT\_WRITE，可读可写

flags：MAP\_SHARED固定

fd：文件描述符

offset：偏移量，一般给0

返回值：

成功：内存映射的起始地址

失败：NULL