**嵌入式是什么？**

嵌入式是一门软件硬件相结合的技术，软件指的是代码/app，硬件指的是嵌入式硬件平台。把app/应用程序移植到嵌入式硬件平台时，在硬件平台中运行该app/应用程序，得到预期的效果，那么这个过程就是嵌入式技术

**一、如何永久保存数据**

我们之前使用数据类型存储数据的时候，一旦整个应用程序结束之后，再次启动该程序，数据都会被还原清零。因为使用数据结构存储的数据是存储在虚拟内存空间中，当程序结束之后，整个虚拟内存空间都会被释放，数据就会被清零。那么，如何才能使数据“永久”保存呢，答案是使用文件存储数据。所以需要在编程中针对文件进行操作（输入和输出，简称IO）。

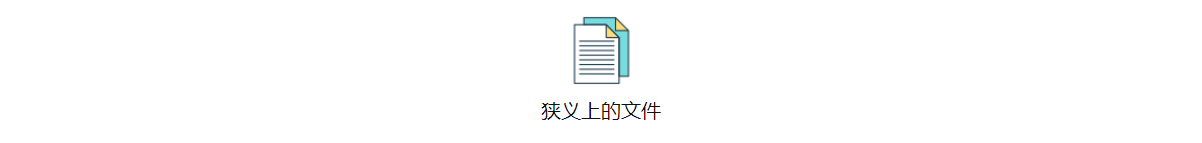
int n = 10;

**open read write close**

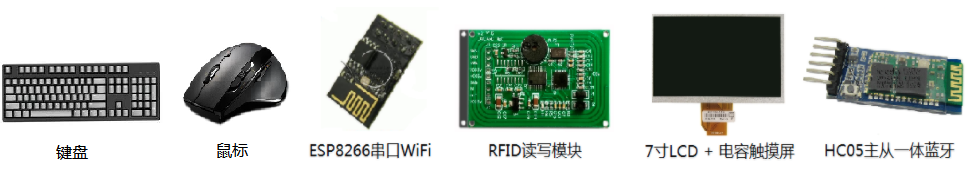
**二、文件的概念**

在Linux系统语境下，文件（file）一般有两个基本含义：

狭义：指普通的文本文件，或二进制文件。包括日常所见的源代码、word文档、压缩包、图片、视频文件等等。



广义：除了狭义上的文件外，几乎所有可操作的设备或接口都可视为文件。包括键盘、鼠标、硬盘、串口、触摸屏、显示器等，也包括网络通讯端口、进程间通讯管道等抽象概念。



**在linux下，一切都是文件。**
除了我们平时常见文件：1.txt/2.jpg/3.mp3..是文件之外，linux系统还会把硬件设备当作是文件，例如：LED灯、触摸屏、LCD液晶屏幕，蜂鸣器等等，这些硬件设备在linux的眼中，都是文件。
这句话是站在内核的角度说的，因为在内核中所有的设备（除了网络接口）都一律使用 Linux 独有的虚拟文件系统（VFS） 来管理。这样做的最终目的，是将各种不同的设备用“文件”这个概念加以封装和屏蔽，简化应用层编程的难度。

**文件IO的概念就是对文件进行输入（写）输出（读）操作**

**三、Linux系统中文件的分类**

在Linux中，文件总共被分成了7种，他们分别是：

普通文件：存在于外部存储器中，用于存储普通数据。

目录文件：用于存放目录项，是文件系统管理的重要文件类型。

管道文件：一种用于进程间通信的特殊文件，也称为命名管道FIFO。

套接字文件：一种用于网络间通信的特殊文件。

链接文件：用于间接访问另外一个目标文件，相当于Windows快捷方式。

字符设备文件：字符设备在应用层的访问接口。

块设备文件：块设备在应用层的访问接口。

gec@ubuntu:~$ ls -l

-rw-r--r-- 1 gec gec 345 Sep 12:38 a.zip

drwxr-xr-x 2 gec gec 1024 Sep 12:38 dir/

prw-r--r-- 1 gec gec 0 Sep 12:38 pipe

srw-r--r-- 1 gec gec 0 Sep 12:38 socket

lrw-r--r-- 1 gec gec 4 Sep 12:38 link -> a.zip

crw-r--r-- 1 gec gec 1, 3 Sep 12:38 character

brw-r--r-- 1 gec gec 5, 1 Sep 12:38 block

注意到，每个文件信息的最左边一栏，是各种文件类型的缩写，从上到下依次是：

-（regular）普通文件

d（directory）目录文件

p（pipe）管道文件（命名管道）

s（socket）套接字文件（Unix域/本地域套接字）

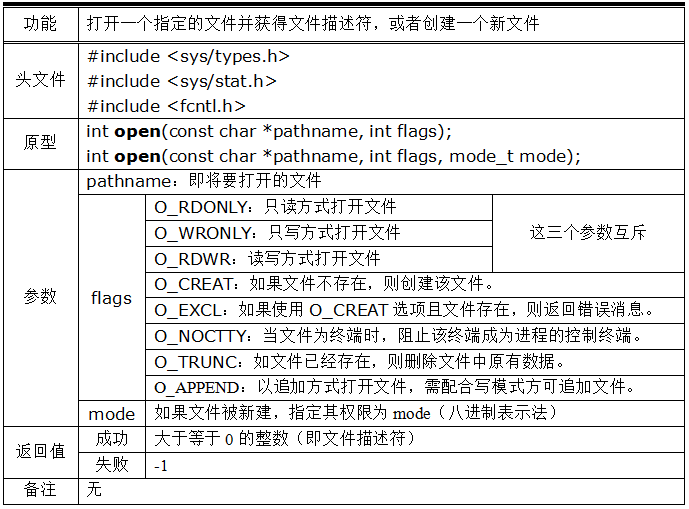
l（link）链接文件（软链接）

c（character）字符设备文件

b（block）块设备文件

**四、系统IO基本API**

**1、打开文件 man 2 open**



注意：open函数什么时候会打开失败？

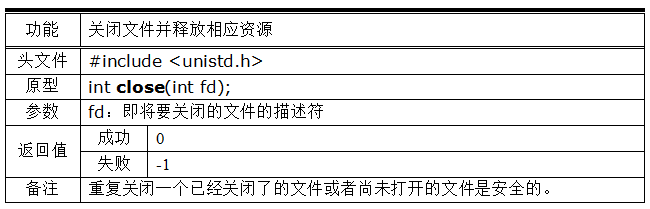
1）你打开的路径名不存在时。

2）如果文件本身的权限不允许，那么操作权限不对，也会失败。

例如：文件本身的权限"-wx-wx-wx",如果以O\_RDONLY / O\_RDWR去打开文件，那么就会失败。

（记得在家目录下才能去验证，共享目录下文件的权限没法改变）

**2、关闭文件 man 2 close**



例子： 访问家目录下test.txt，如果访问成功，则输出"open file success",否则，输入"open file error",并关闭文件，如果关闭成功，则输出"close file success",否则输出"close file error"。

#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int main()

{

    int fd = open("./test.txt",O\_RDONLY);

    if(fd == -1){

        perror("open file error");

        return -1;

    } else{

        printf("open file success\n");

    }

    close(fd);

}

练习1： 看看文件描述值是多少？

练习2：验证 在程序运行过程中，如果每次打开文件不关闭，最多能打开多少次

int main()

{

    int fd;

    while(1)

    {

        fd = open("./test.txt",O\_RDONLY);

        if(fd == -1){

            perror("open file error");

            break;

        }

        printf("fd:%d\n",fd);

    }

    close(fd);

}

范围：0~1023

记住：文件描述符的资源有限，打开文件后，一定要记得关闭。

**五、文件描述符本质**

**1、文件描述符概念**

文件描述符是open函数的返回值，当open()执行成功后，就会返回一个非负的，最小的，没有使用过的整数。

例如：

3 = open("1.txt"); //后面的代码中，3就是代表1.txt这个文件。

4 = open("2.txt"); //后面的代码中，4就是代表2.txt这个文件。

结论：将来处理文件时，我们不需要提供文件名字，只需要提供文件对应的文件描述符就可以。

**2、标准输入/输出/出错**

访问文件时，发现fd从3开始分配，说明0/1/2已经被占用，究竟是谁使用了呢？其实在程序执行的时候，就会默认打开3个文件，分别是"标准输入"、"标准输出"、"标准出错"，他们其实是一个宏定义来的，是被定义在一个头文件中，头文件路径：/usr/include/unistd.h

/\* Standard file descriptors. \*/

#defineSTDIN\_FILENO0//标准输入设备文件 -> 对象：键盘 -> 控制键盘输入

#defineSTDOUT\_FILENO1//标准输出设备文件 -> 对象：屏幕 -> 控制屏幕显示内容

#defineSTDERR\_FILENO2//标准出错设备文件 -> 对象：屏幕 -> 控制屏幕显示错误

可以理解： 程序刚开始执行的时候，0=open("标准输入");

**3、举例子**

验证文件描述符特性：非负、最小、没有使用过、整数。

假设当前目录下有a.txt、b.txt、c.txt、d.txt。

open("a.txt"); //3

open("b.txt"); //4

open("c.txt"); //5

关闭b.txt //---4

open("d.txt"); //4

关闭a.txt //---3

关闭c.txt //5

open("a.txt") //3

open("c.txt"); //5

关闭d.txt //4

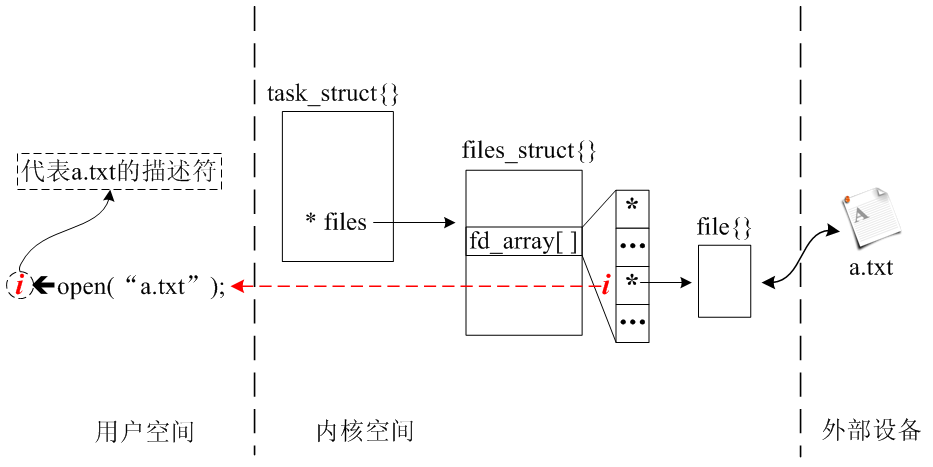
请问open("d.txt") -> 4

结论： 打开一个文件，得到一个文件描述符 ---> 申请资源

关闭一个文件，这个文件对应的文件描述符可以被别人使用 ---> 释放资源

**4、本质**

函数 open() 的返回值，是一个整型 int 数据。这个整型数据，实际上是内核中的一个称为 fd\_array 的数组的下标：



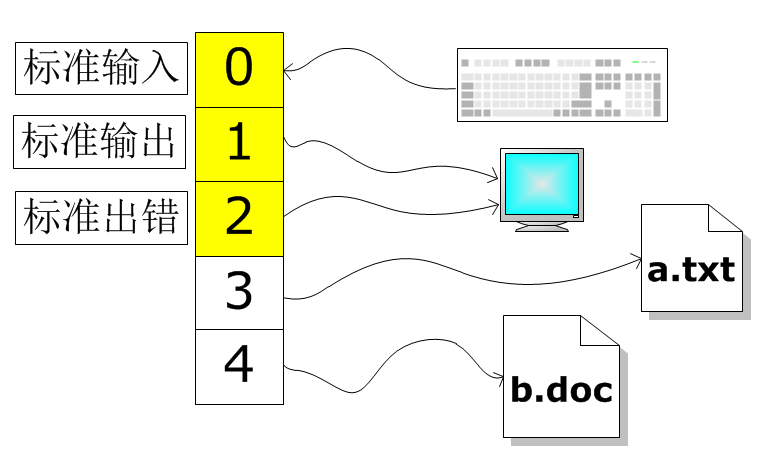
打开文件时，内核产生一个指向 file{} 的指针，并将该指针放入一个位于 file\_struct{} 中的数组 fd\_array[] 中，而该指针所在数组的下标，就被 open() 返回给用户，用户把这个数组下标称为文件描述符，如上图所示。

结论：

文件描述符从0开始，每打开一个文件，就产生一个新的文件描述符。

可以重复打开同一个文件，每次打开文件都会使内核产生系列结构体，并得到不同的文件描述符

由于系统在每个进程开始运行时，都默认打开了一次键盘、两次屏幕，因此0、1、2描述符分别代表标准输入、标准输出和标准出错三个文件（两个硬件）。



**六、文件数据输出/输入**

**1、如何读取文件的数据？ -> read() -> man 2 read**

功能：read - read from a file descriptor

//读取一个文件描述符对应文件的数据

头文件

#include <unistd.h>

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

参数：

fd： 想读取的那个文件的文件描述符

buf： 数据缓冲区(存储读取到的数据)

count：想读取的字节数 （愿望值）

返回值：

返回值 >0 表示成功读取到的字节数

返回值 ==0 表示文件读取到末尾了

返回值 ==-1 读取失败

例子1： 尝试从文本中读取一些数据出来。

#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int main()

{

    int fd = open("./3.txt",O\_RDWR);

    if(fd == -1)

    {

        perror("open error");

    }

    char buf[1024]={0};

    int ret = read(fd,buf,sizeof(buf)-1);

    printf("ret:%d buf:%s\n",ret,buf);

    close(fd);

}

练习3：重复读取一个文件，那么第二次读取的时候是在第一次基础上继续读，还是重新读？ -> 继续读

文件：

aaaaaaaaaa

aaaaaaaaaa

aaaaaaaaaa

bbbbbbbbbb

bbbbbbbbbb

bbbbbbbbbb

int main()

{

    int fd = open("./3.txt",O\_RDWR);

    if(fd == -1)

    {

        perror("open error");

return -1;

    }

    while(1)

    {

        char buf[6]={0};

        int ret = read(fd,buf,sizeof(buf)-1);

        if(ret == 0)//读取完毕

            break;

        printf("ret:%d buf:%s\n",ret,buf);

        //sleep(1);

    }

    close(fd);

}

总结：

1）、如果一个文件很大，可以写一个循环每次获取指定的字节数，当read返回值 为0时，则表示这个文件读完了，则退出

2）、文本换行符是 \r\n ,占两个字节

**2、写入数据到文件中。 -> write() -> man 2 write**

write - write to a file descriptor

//写入数据到文件描述符中

#include <unistd.h>

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

参数：

fd: 想写入的文件的文件描述符

buf: 数据缓冲区（存放写入到文件的数据）

count: 写入的字节数

返回值：

成功：已经写入的字节数

失败：-1

例子2：尝试写入数据到空白的文件中。

#include<stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int main()

{

    int fd = open("./3.txt",O\_RDWR|O\_TRUNC|O\_CREAT,0777);

    if(fd == -1)

    {

        perror("open error");

        return -1;

    }

    char buf[] = "hello world";

    write(fd,buf,strlen(buf));//有多少写多少

    close(fd);

}

思考：如果写入的字节数比真实的数据要大，会发生什么情况？

    int fd = open("./3.txt",O\_RDWR|O\_TRUNC|O\_CREAT,0777);

    if(fd == -1)

    {

        perror("open error");

        return -1;

    }

    char buf[] = "hello world";

    write(fd,buf,1024);//会乱码

总结：有多少字节的数据，count就写多少，如果count比真实数据大，那么文本的后面可能有乱码。（文本不够，乱码来凑）

练习4：重复写入一个文件，那么第二次写入的时候是在第一次基础上继续写，还是重新写？ -> 继续写

练习5：先读一些数据，然后再写，那么写的时候是重新写，还是在读取完的那个位置继续写？-> 继续写

结论：open()打开文件时，文件定位都是在最开头的，文件定位随着读取/写入字节而往后偏移。

**七、文件偏移量**

1、 什么是文件偏移量？

文件偏移量就是文件当前的光标位置，默认打开文件时，文件的光标位置都是在最开头的。

2、怎样才能使得文件偏移量发生改变？

1）使用读写函数可以使得偏移量发生变化。

fd = open("test.txt"); 偏移量：0

write(fd,"hello",5); 偏移量：5

2）如何使得不调用读写函数前提下进行偏移？ -> lseek() -> man 2 lseek

功能：lseek - reposition read/write file offset

//重新定位读取/写入的偏移量

#include <sys/types.h>

#include <unistd.h>

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

参数：

fd：需要发生偏移的那个文件的文件描述符

offset： 基于光标基准点偏移的字节数 [+](往后偏移) -(往前偏移)

whence：光标基准点

SEEK\_SET： 相对于文件开头

SEEK\_CUR： 相对于当前的位置

SEEK\_END： 相对于文件的末尾

返回值：

成功：光标距离文件开头的偏移量（字节）。

失败：-1。

例子： 验证lseek的参数。

int main(int argc,char \*\*argv)

{

int fd = open("./1.txt",O\_RDWR); //文件内容：hello world

if(fd == -1)

{

printf("open file error\n");

return -1;

}

//基于文件开头向后偏移5个字节

int ret = lseek(fd,5,SEEK\_SET);

printf("ret:%d\n",ret);

//字符串123 会直接从光标5的位置直接写入，会覆盖原来的数据

write(fd,"123",3);

close(fd);

}

思考：

1、如果当前已经在开头了，还往前偏移会怎样？

lseek()函数会执行失败，返回-1，当前文件的定位还是在开头。

int fd = open("./1.txt",O\_RDONLY);

if(fd == -1)

{

printf("open file error\n");

return -1;

}

int ret = lseek(fd,-10,SEEK\_SET);

printf("ret:%d\n",ret); //ret == -1

2、如果当前已经在文件的末尾了，还往后偏移会怎样？

会发生偏移，但是在windows看的时候，还是在文件的末尾。

int main(int argc,char \*\*argv)

{

int fd = open("./1.txt",O\_RDWR); //文件内容：hello world

if(fd == -1)

{

printf("open file error\n");

return -1;

}

//基于文件末尾 向后偏移10个字节

int ret = lseek(fd,10,SEEK\_END);

printf("ret:%d\n",ret); //偏移成功 ret == 21

//如果此时不写入数据，直接关闭文件，文件的大小还是原来的11字节

write(fd,"123",3); //此时文件大小为24个字节

close(fd);

}

特性：lseek函数不仅可以用来调整当前文件偏移量，而且还可以将文件位置偏移到文件之外，形成一个空洞，这种特性其实是非常重要的，它提供了可以在不同地方同时写一个文件的可能，对于一个较大的文件我们可以在文件中定位到一个指定的地方，让多个进程同时在不同的偏移量处写入文件数据。相当于网络中的多点下载。

练习1： 自己修改lseek.c 调用write 之后，偏移到开头 ，再读

练习2：直接读取3.txt文件里面的"hello world good" 中的 "good"

思路：

1》打开3.txt

2》基于文件开头向后偏移12个字节

3》读取4个字节数据放到缓冲区

4》输出数据

5》关闭文件

答案：

练习2：

int getfileSize(int fd)

{

//1、先保存当前文件的位置

int curPos = lseek(fd,0,SEEK\_CUR);

//2、偏移文件到末尾，返回值就是文件末尾到开头的字节数（也就是文件的大小）

int fileSize = lseek(fd,0,SEEK\_END);

//3、恢复原来的位置

lseek(fd,curPos,SEEK\_SET);

//4、返回文件的大小

return fileSize;

}

练习3：

int main(int argc,char \*\*argv)

{

//1、打开3.txt

int fd = open("./3.txt",O\_RDWR);

if(fd == -1)

{

printf("open file error\n");

return -1;

}

//2、基于文件开头向后偏移12个字节

lseek(fd,12,SEEK\_SET);

//3、读取4个字节的数据放到缓冲区

char buf[5]={0};

read(fd,buf,4);

//4、输出数据

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

//5、关闭文件

close(fd);

}