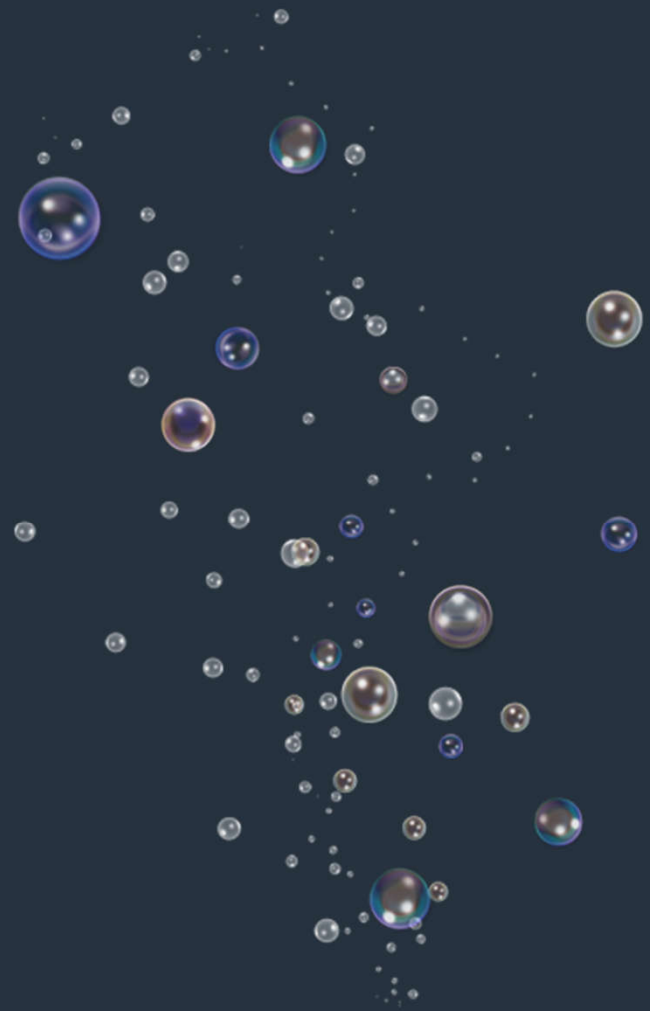


文件操作

大连理工大学 赖晓晨

对文件的操作有两种方式

- 系统调用
  - 系统调用可以直接访问内核提供的丰富功能，是内核的低级接口
  - 系统调用存在于内核空间，使用不当也许会损坏系统
  - 对文件的操作使用文件描述符：整数
- 库函数
  - 库调用属于Linux编程接口中的较高层接口
  - 库函数存在于用户空间，对系统危险较小
  - 对文件的操作使用文件指针：FILE*
- 实际上很多库函数都是利用系统调用来实现的

打开文件的函数

- 打开文件：

```
#include <stdio.h>
FILE* fopen(const char* path, const char* mode)
```

返回值：指向文件的指针；如果出现错误，返回NULL并设置errno变量，如果文件不存在，则以666权限创建此文件

path：文件的路径

mode：文件的打开方式

文件的打开方式

- r：只读
- r+：读写
- w：只写
- w+：读写
- a：只追加
- a+：读、追加

关闭文件的函数

关闭文件：

```
#include <stdio.h>
int fclose(FILE* stream)
```

- 返回值：关闭成功返回0，否则返回EOF
- 参数：打开文件时返回的指针
- 说明：一旦文件被关闭，任何对这个文件的访问包括fclose调用都会导致不可预料的错误

读写文件

- 对文件的读写操作使用函数fread和fwrite
- 函数fread和fwrite允许从文件流读出数据或者向文件流写入数据

读文件

fread原型如下:

```
#include <stdio.h>
size_t  fread(void* ptr, size_t size, size_t n, FILE* pf)
```

- ptr指向保存从文件中读到信息的缓冲区
- size保存读取的每一个“记录”的大小
- n保存读取的“记录”数
- pf指针指向要读取的文件流
- 返回实际读到的“记录”数

读文件

fwrite原型如下:

```
#include <stdio.h>
size_t  fwrite(void* ptr, size_t size, size_t n, FILE* pf)
```

- ptr指向保存要写入到文件中的信息的缓冲区
- size保存写入的每一个“记录”的大小
- n保存写入的“记录”数
- pf指针指向要写入的文件流
- 返回实际写入的“记录”数

获取文件状态

```
#include <stdio.h>

//检查是否已到文件尾，是则返回非零值
int feof(FILE* s);
//如果文件流出错则返回非零值，但是不设置errno变量
int ferror(FILE* s);
//清除在文件上已经设置的错误位
void clearerr(FILE* s);
//返回与给定文件流相关联的文件描述符
int fileno(FILE* s);
```

文件定位

- 文件定位函数

```
#include <stdio.h>
int fseek(FILE* s, long offset, int whence);
```

- fseek函数把当前位置设置到s指向的文件的offset处，参数whence可以是SEEK_SET、SEEK_CUR、SEEK_END，决定了以文件的起始、文件的当前位置或者文件的结尾来计算offset
- 正常情况返回相对起始位置的偏移，出错返回-1

删除文件

- 删除文件函数

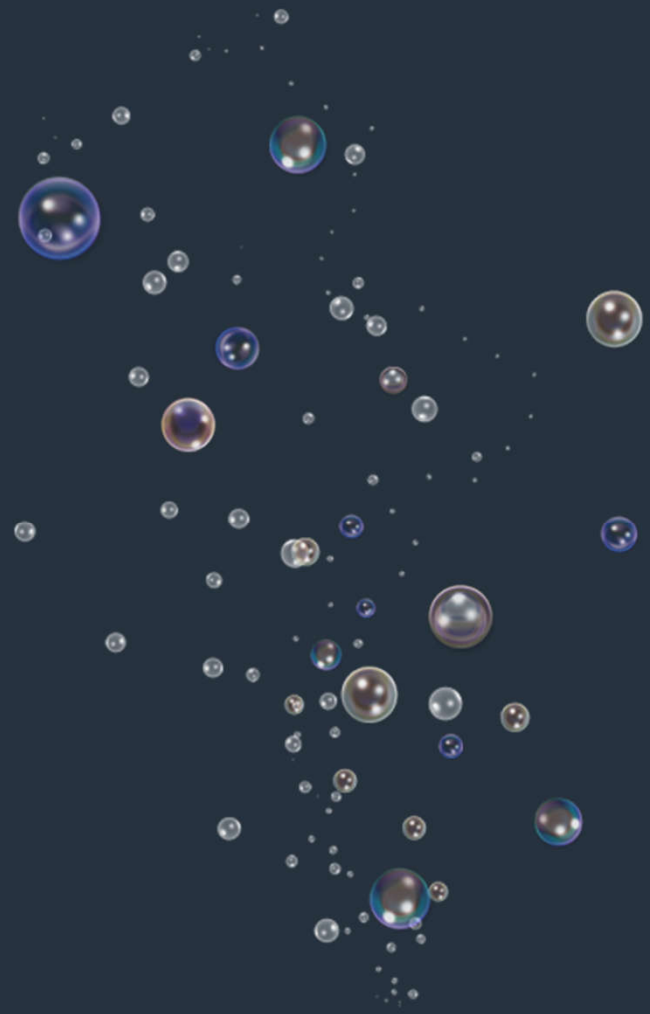
```
#include <stdio.h>
int remove( const char* pathname );
```

- 文件改名函数

执行成功返回**0**，否则**-1**

```
#include <stdio.h>
int rename( const char* oldpath,
   const char* newpath );
```

执行成功返回**0**，否则**-1**



嵌入式软件设计

大连理工大学 赖晓晨