标准I/O(复习)

回顾

基本概念

- 库、API、系统调用。
- 特殊的文件：stdin, stdout, stderr。
- I/O缓存。
- 文件操作相关的API。

回顾

编程相关函数

- 打开文件fopen。
- 关闭文件fclose。
- 读文件fgetc/fgets/fread。
- 写文件fputc/fputs/fwrite。
- 定位操作指针位置fseek，ftell
- 判断文件结束feof。

标准I/O - fopen() - mode参数

打开一个标准I/O流的六种不同的方式

限　制	r	w	a	r+	w+	a+
文件必须已存在	•			•		
擦除文件以前的内容		•			•	
流可以读	•			•	•	•
流可以写		•	•	•	•	•
流只可在尾端处写			•			•

access

- #include <unistd.h>
- int access(const char *pathname, int mode);
- 说明：检查指定的文件是否存在，并具有权限
  - pathname：文件的路径
  - mode：检查的权限，可取如下值

R_OK	W_OK	X_OK	F_OK
可读权限	可写权限	可执行权限	读写执行权限

- 返回值
  - 成功：返回0
  - 失败：返回-1

tmpfile

- #include <stdio.h>
- FILE *tmpfile(void);

- 说明：
  - tmpfile() 函数会以read/write (w+b)的模式创建一个临时文件，该临时文件在用户调用fclose或者程序结束时会被删除。

- 返回值：
  - 成功：返回一个可读写的文件指针
  - 失败：返回NULL

truncate

- #include <unistd.h>
- #include <sys/types.h>
- int truncate(const char *path, off_t length);

- 说明：truncate函数将文件长度截短或接长。
  - 第一参数是文件名。
  - 第二个参数是设置后的文件长度，如果文件被截短，截掉的内容丢失，如果接长，后面补0。
- 返回值：
  - 成功：返回0
  - 失败：返回-1

strerror

- #include <string.h>
- char *strerror(int errnum);

- 将错误码转化成解释性的字符串
- 用法：
  - printf(“xxx: %s”, strerror(errno));
- 类似于perror(“xxx”);

文件I/O

文件I/O – 介绍

嵌入式学院
华清远见旗下品牌

文件I/O

① 不带缓冲
② 通过文件描述符来访问文件
③ 属于系统调用

文件I/O常用函数

① open()/creat()
② close()
③ read()
④ write()
⑤ lseek()

课程目标

- 1、理解并熟练使用文件I/O相关函数。
- 2、使用文件I/O完成一个键盘记录程序，将键盘的敲击动作录制成文件。
- 3、使用文件I/O将键盘录制的文件播放出来，完成用户的模拟输入。

文件描述符

概念：文件流指针，文件描述符

文件流指针：C库中用于操作某个文件的指针变量，指向一个C库的文件结构。

文件描述符：内核中一个打开文件的唯一标识符。文件描述符是一个**非负整数**。当打开一个现存文件或创建一个新文件时，内核向进程返回一个文件描述符，进程通过该描述符操作文件。

open

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

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

头文件：(man 2 open)

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

pathname ： 要打开文件的路径（相对路径、绝对路径）

flags ： 打开标识，包含打开权限

mode ： （可选），如果是创建文件，标识文件的权限

成功：

返回：文件描述符

失败：-1

open

open()调用返回的文件描述符一定是**最小的未用描述符数字**。

open()可以打开或者创建一个常规文件，可以打开设备文件，但是不能创建设备文件。

文件I/O – 标准I/O

标准I/O预定义3个流，文件I/O 预打开3个文件描述符

标准输入	0	STDIN_FILENO	stdin
标准输出	1	STDOUT_FILENO	stdout
标准错误输出	2	STDERR_FILENO	stderr

STDIN_FILENO、STDOUT_FILENO、STDERR_FILENO。定义在头文件<unistd.h>中。

文件I/O – open()

<table>
<tr><td>原型</td><td colspan="3">int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);</td></tr>
<tr><td rowspan="8">参数</td><td>pathname</td><td colspan="2">被打开的文件名（可包括路径名）。</td></tr>
<tr><td rowspan="7">flags</td><td>O_RDONLY：只读方式打开文件。</td><td rowspan="3">这三个参数互斥</td></tr>
<tr><td>O_WRONLY：可写方式打开文件。</td></tr>
<tr><td>O_RDWR：读写方式打开文件。</td></tr>
<tr><td colspan="2">O_CREAT：如果该文件不存在，就创建一个新的文件，并用第三的参数为其设置权限。</td></tr>
<tr><td colspan="2">O_EXCL：与O_CREAT同时使用，文件存在可返回错误消息。</td></tr>
<tr><td colspan="2">O_TRUNC：如文件已经存在，那么打开文件时先删除文件中原有数据。</td></tr>
<tr><td colspan="2">O_APPEND：以添加方式打开文件，所以对文件的写操作都在文件的末尾进行。</td></tr>
<tr><td>mode</td><td colspan="2">如果是create文件，指明文件的存取权限，为8进制表示法。</td></tr>
</table>

文件I/O – close()

调用close()函数可以关闭一个打开的文件。

```
#include <unistd.h>
int close(int fildes);
```

调用成功返回0，出错返回-1，并设置errno。

当一个进程终止时，该进程打开的所有文件都由内核自动关闭。

实验1

1、以所有者只写方式(参数O_WRONLY)打开文件(mode=0644)，如果文件不存在则创建(参数O_CREAT)，如果文件存在则清空原来内容(参数O_TRUNC)：
2、判断文件dir /proc/scsi/usb-storage是否存在，不存在则等待，直到存在时才退出。

思考：fopen中mode和open中flags的对应关系？

权限掩码

fopen()没有设定创建文件权限的参数，默认具有如下权限(0666或者-rw-rw-rw)：

S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH

open()中mode标识了创建文件时，文件的访问权限；但文件的实际访问权限要减去文件权限掩码。

文件权限掩码：文件权限中要被掩（阉）掉的权限位。用户可以通过umask修改文件存取的权限，其结果为(0666 & ～umask)，（如果umask=022，open中指定mode为666，则实际的权限为644）。

实验2

- 1、创建一个文件，权限为只有owner用户有权限读写，其他用户无任何权限。

read/write


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

功能：读文件内容到buf，或将buf内容写入文件。

头文件：#include <unistd.h>

fd：　　要读写文件的路径（相对路径、绝对路径）

buf：　　要读入或写出的缓冲区，提前申请好的

count ：要读入或写出的字节数

成功：

返回：实际读写的字节数

失败：-1

read/write注意

read() 调用成功返回读取的字节数，如果返回小于期望值，表示到达文件末尾，如果返回-1，表示出错。

write() 调用成功返回的写入的字节数可能小于期望写入的字节数，在一些固定容量的特殊文件里面（如管道）会出现，这是需要循环写入文件。

buf 参数需要有调用者来分配内存，并在使用后，由调用者释放分配的内存。

read返回的读出长度即是buf有效数据的长度，buf中下一个字符是否为0和读没有任何关系。（之前学的fgets才会将最后一个字符置0）

思考：C库里面的fread，fwrite用read，write是如何实现的？

实验3

1、将数组{0x0, 0x1, 0x2, 0x3, 0x04, …. 0xff }的数据写入文件：

程序设计

- ✓ 初始化一段缓存区，为缓冲区循环赋值。
- ✓ 以可写O_WRONLY | O_TRUNC | O_CREAT 方式打开文件。
- ✓ 将缓冲区中的内容写入文件。
- ✓ 判断是否写入完成，是则退出。

2、将上面程序生成的文件读出来，并将第48到57个字符printf

3、打开一个新的终端tty，将2中写入文件的内容写入新的终端tty。

lseek

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

功能：调整或得到文件当前读写的位置。

头文件：

#include <unistd.h>

#include <sys/types.h>

fd：　　用open打开的文件描述符。

offset：　想要调整的文件读写位置的相对偏移

whence:　调整文件读写位置的基准。

SEEK_SET：当前位置为文件的开头

SEEK_CUR：当前位置为文件指针的位置

SEEK_END：当前位置为文件的结尾

返回值

成功：文件的当前位移

出错：　-1

lseek

原型	off_t lseek(int fd,off_t offset,int whence);	
参数	fd：文件描述符。	
	offset：偏移量，每一读写操作所需要移动的距离，单位是字节的数量，可正可负（向前移，向后移）	
	whence(当前位置基点)：	SEEK_SET：当前位置为文件的开头，新位置为偏移量的大小。
		SEEK_CUR：当前位置为文件指针的位置，新位置为当前位置加上偏移量。
		SEEK_END：当前位置为文件的结尾，新位置为文件的大小加上偏移量的大小。
返回值	成功：文件的当前位移	
	-1：出错	

文件类型

嵌入式学院
华清远见旗下品牌

- 概念：
  - 常规文件
  - 管道文件
  - socket
  - 块设备
  - 字符设备
  - 文件类型和文件打开类型

文件I/O – lseek()

- ✓ 每个打开的文件都有一个与其相关的“当前文件位移量”，它是一个非负整数，用以度量从文件开始处计算的字节数。
- ✓ lseek()只对常规文件有效，对socket、管道、FIFO等进行lseek()操作失败。
- ✓ lseek()仅将当前文件的位移量记录在内核中，它并不引起任何I/O操作。
- ✓ 文件位移量可以大于文件的当前长度，在这种情况下，对该文件的写操作会延长文件，并形成空洞。
- ✓ lseek成功返回当前的文件偏移，fseek成功返回0，不要混淆。

实验4

嵌入式学院
华清远见旗下品牌

- 1、用lseek得到文件大小。
- 2、读出.pdf，png，linux可执行文件的第1～4个字符并打印。

- 思考：用文件IO复制文件或读文件时，如何判断文件结束了？

一些常用的文件

- /dev/null
- /dev/zero
- /dev/input/event*
- /dev/pts
- /dev/random，/dev/urandom

键盘输入文件

- /dev/input/event0（也可能是event1）
- 每一次按键都会触发两个事件，键按下，键弹起，如果长按，还会触发长按的消息。
- 每一个消息都会写到字符文件/dev/input/event0中，结构为struct input_event（结构体定义在linux/input.h中）。

实验5

嵌入式学院
华清远见旗下品牌

- 1、录制30秒的键盘输入，写成文件保存为常规文件key_record.log
- 2、将key_record.log按结构读出来，并写入键盘字符文件，模拟用户输入。

标准I/O和文件IO区别

I/O模型	文件I/O	标准I/O
缓冲方式	非缓冲I/O	缓冲I/O
操作对象	文件描述符	流(FILE *)
打开	open()	fopen()/freopen()/fdopen()
读	read()	fread()/fgetc()/fgets()…
写	write()	fwrite()/fputc()/fputs()…
定位	lseek()	fseek()/ftell()
关闭	close()	fclose()

文件和目录

课程目标

掌握文件属性获取与修改相关的操作函数

stat()fstat()/lstat()

掌握目录操作相关的函数

chdir()/opendir()/readdir()…

stat

int stat(const char *path, struct stat *buf);

功能：获取文件的属性信息，保存到指定的struct stat结构体中。

path：文件的路径。

buf： struct stat结构体指针

返回值：

成功：0

失败：-1

struct stat

```
struct stat {
        dev_t st_dev;     /* ID of device containing file */
        ino_t  st_ino;    /* inode number */
        mode_t            st_mode;    /* protection */
        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 */
        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 */
};
```

mode_t st_mode

S_IFSOCK	0140000	socket
S_IFLNK	0120000	符号连接
S_IFREG	0100000	常规文件
S_IFBLK	0060000	块设备
S_IFDIR	0040000	目录
S_IFCHR	0020000	字符设备
S_IFIFO	0010000	管道

S_IRUSR	00400	所有者有可read
S_IWUSR	00200	所有者有可write
S_IXUSR	00100	所有者有可execute
S_IRGRP	00040	所有者的group可 read
S_IWGRP	00020	所有者的group可write
S_IXGRP	00010	所有者的group可execute
S_IROTH	00004	其他用户可read
S_IWOTH	00002	其他用户可write
S_IXOTH	00001	其他用户可execute

mode_t st_mode

宏定义：

S_ISREG(m)	判断是否是否为常规文件
S_ISDIR(m)	判断是否为目录文件
S_ISCHR(m)	判断是否为字符文件
S_ISBLK(m)	判断是否为块文件
S_ISFIFO(m)	判断是否为管道文件
S_ISLNK(m)	判断是否为链接文件
S_ISSOCK(m)	判断是否为SOCKET文件

文件类型

目录文件(directory file) 这种文件包含了其他文件的名字以及指向与这些文件有关信息的指针。对一个目录文件具有读许可权的任一进程都可以读该目录的内容，但只有内核可以写目录文件。

普通文件(regular file) 这是最常见的文件类型，这种文件包含了某种形式的数据。至于这种数据是文本还是二进制数据对于内核而言并无区别。对普通文件内容的解释由处理该文件的应用程序进行。

字符特殊文件(character special file) 这种文件用于系统中某些类型的设备。

块特殊文件(block special file) 这种文件典型地用于磁盘设备。系统中的所有设备或者是字符特殊文件，或者是块特殊文件。

FIFO 这种文件用于进程间的通信，有时也将其称为命名管道。

套接口(socket) 这种文件用于进程间的网络通信。套接口也可用于在一台宿主机上的进程之间的非网络通信。

符号连接(symbolic link) 这种文件指向另一个文件。

实验6

- 1、用stat函数得到文件的大小，mod时间。
- 2、用stat函数得到文件的类型，只区分常规文件、目录文件,others等。

linux文件属性详细说明：
http://www.cnblogs.com/Theladyflower/archive/2011/07/26/2117102.html

目录操作（自学）

嵌入式学院
华清远见旗下品牌

- chdir()/opendir()/readdir()…

- #include <sys/types.h>
- #include <dirent.h>

- DIR *opendir(const char *name);
- struct dirent *readdir(DIR *dirp);

struct dirent

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

实验7

嵌入式学院
华清远见旗下品牌

- 1、完成一个目录列举的程序，输出目录下的所有文件。
- 2、完成一个复制目录的程序，可先不考虑子目录。
- 3、完成一个复制目录的程序，可先不考虑子目录，新文件保持原来的权限（不同用户读写执行的权限）。

静态库和动态库

概念理解

静态库：是一个提供API函数的编译环境，有了静态库，就可以使用静态库中的函数编写应用程序，把函数的汇编代码放入应用程序实体。

动态库：是一个应用程序的运行环境，有了动态库，就不需要链接编译函数代码，只要把动态库加载入内存，找到函数执行的地址，执行函数的汇编代码即可。

库无处不在

除了部分汇编语言开发，每一种语言都有自己的开发库（lib）和运行库 （runtime），比如：

Python运行库

Java运行库

C++库

C库（Linux系统是用c写的，所以自带c的运行库环境）

创建一个静态库

- 将自己工作中常用的函数封装起来，以后使用他们就像使用printf一样方便。
- 1、写函数实体， fun.c
- 2、写函数头文件，fun.h
- 3、编译生成.o文件。
  - gcc –c fun.c
- 4、将.o文件链接成静态库
  - ar cr libfun.a fun.o
- 5、将头文件fun.h和静态库libfun.a发布出去。
- 6、你的粉丝使用你的库，-L. -lfun

更多：http://blog.csdn.net/kkk8000/article/details/6016525

创建一个动态库

- 体会一下**的感觉
- 1、写函数实体， fun.c
- 2、写函数头文件，fun.h
- 3、编译生成.o文件。
  - gcc –c fun.c
- 4、将.o文件链接成动态库
  - gcc -shared -fPIC -o libfun.so fun.o
- 5、将函数介绍pdf和动态库libfun.so发布出去。
- 6、你的粉丝学习使用你的库，-L. -lfun

扩展学习

- ftruncate
- remove/rmdir
- mkdir
- chdir
- scandir
- chmod/fchmod
- link/unlink
- utime
- rename