文 件 I/O

一口Linux

无线传感器网项目实战

公众号:一口Linux

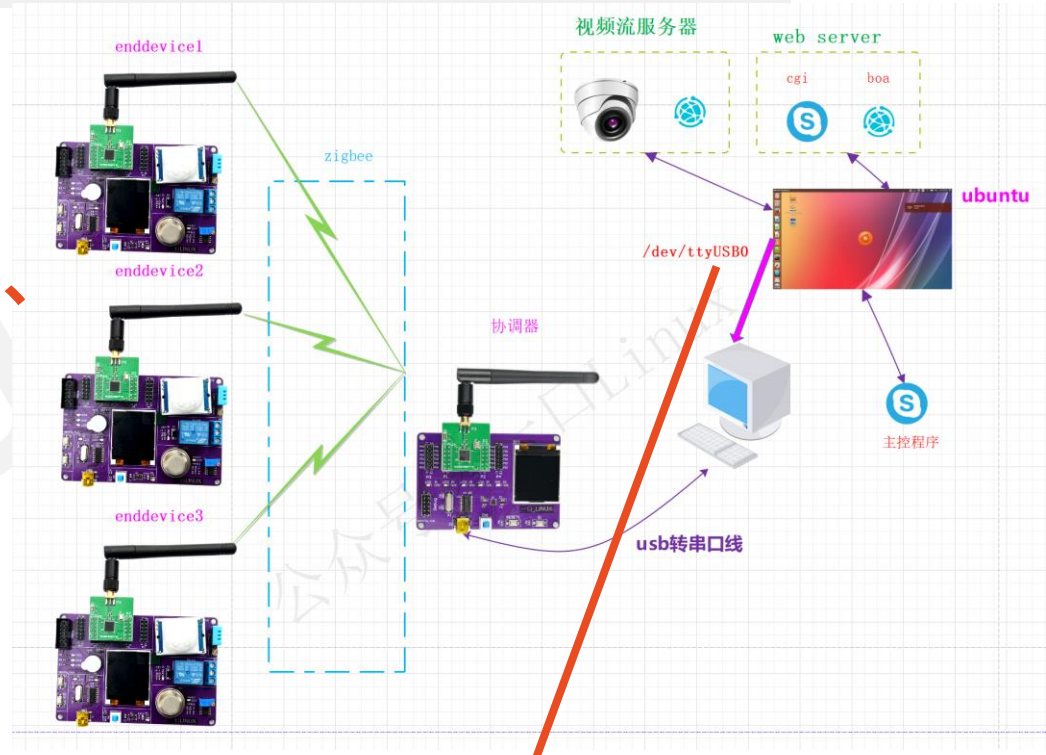
彭老师个人微信号

01

前言

本项目何处用到I/O操作?

- 本项目中协调器通过串口与上位机 通信
- 串口设备被模拟成
  - /dev/ttyUSB0、/dev/ttyUSB1、 /dev/ttyS0、/dev/ttyS1
- 应用程序只需要通过系统调用操作设备文件
  - open()/read()/write()/close()…




```
peng@ubuntu:~/work$ ls /dev/ttyUSB0 -l
crw-rw---- 1 root dialout 188, 0 Apr 24 06:48 /dev/ttyUSB0
```

文件类型
C：字符设备

访问权限

设备号

串口设备文件名

02

文件I/O

什么是文件IO

- 在LINUX/UNIX系统下，有着 **一切皆文件**的定义，
- 所谓文件IO就是指文件的 输入和 输出。

Linux下文件类型

类型	简称	描述
普通文件	-	如mp4、pdf、html log; 用户可以根据访问权限对普通文件进行查看、更改和删除，包括 纯文本文件(ASCII)；二进制文件(binary)；数据格式的文件(data)；各种压缩文件. 第一个属性为 [-]
目录文件	d	/usr/ /home/ 目录文件包含了各自目录下的文件名和指向这些文件的指针，打开目录事实上就是打开目录文件，只要有访问权限，就可以随意访问这些目录下的文件。能用#cd命令进入的。第一个属性为[d]，例如 [drwxrwxrwx]
硬链接	-	若一个inode号对应多个文件名，则称这些文件为硬链接。硬链接就是同一个文件使用了多个别名删除时，只会删除链接， 不会删除文件; 硬链接的局限性:1. 不能引用自身文件系统以外的文件，即不能引用其他分区的文件;2. 无法引用目录;
符号链接（软链接）	l	若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接，克服硬链接的局限性， 类似于快捷方式，使用与硬链接相同。
字符设备文件	c	文件一般隐藏在/dev目录下，在进行设备读取和外设交互时会被使用到 即串行端口的接口设备，例如键盘、鼠标等等。第一个属性为 [c]。 #/dev/tty的属性是 crw-rw-rw-，注意前面第一个字 c，这表示字符设备文件
块设备文件	b	存储数据以供系统存取的接口设备，简单而言就是硬盘。 # /dev/hda1 的属性是 brw-r—- ，注意前面的第一个字符是b，这表示块设备，比如硬盘，光驱等设备 系统中的所有设备要么是块设备文件，要么是字符设备文件，无一例外
FIFO管道文件	p	管道文件主要用于进程间通讯。FIFO解决多个程序同时存取一个文件所造成的错误。比如使用mkfifo命令可以创建一个FIFO文件，启用一个进程A从FIFO文件里读数据，启动进程B往FIFO里写数据，先进先出，随写随读。 # pipe
套接字	s	以启动一个程序来监听客户端的要求，客户端就可以通过套接字来进行数据通信。用于进程间的网络通信，也可以用于本机之间的非网络通信，第一个属性为 [s]，这些文件一般隐藏在/var/run目录下，证明着相关进程的存在

函数介绍

文件I/O介绍

- open()
- close()
- read()
- write()
- lseek()

- **系统调用 打开**(open) read/write close

文件I/O – open()

- 调用open()函数可以打开或者创建一个文件。

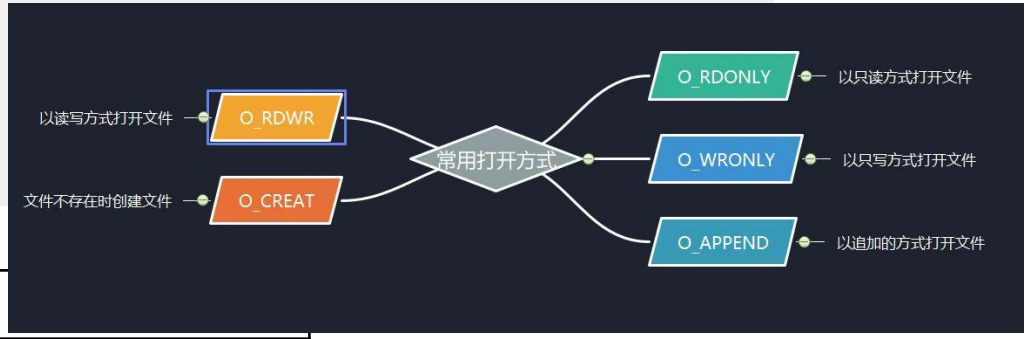
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

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

open() 调用成功返回文件描述符，失败返回-1，并设置errno。

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()可以打开设备文件，但是不能创建设备文件，设备文件必须使用mknod()创建。

文件I/O – open()参数



原型	int open(const char *pathname, int flags, mode_t mode);		
参数	pathname	被打开的文件名（可包括路径名）。	
	flags	O_RDONLY：只读方式打开文件。	这三个参数互斥
		O_WRONLY：可写方式打开文件。	
		O_RDWR：读写方式打开文件。	
		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。	
		O_EXCL：如果使用O_CREAT时文件存在，则可返回错误消息。这一参数可测试文件是否存在。	
		O_NOCTTY：使用本参数时，如文件为终端，那么终端不可以作为调用open()系统调用的那个进程的控制终端。	
		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。	
		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。	
	mode	被打开文件的存取权限，为8进制表示法。	

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 fd);
```

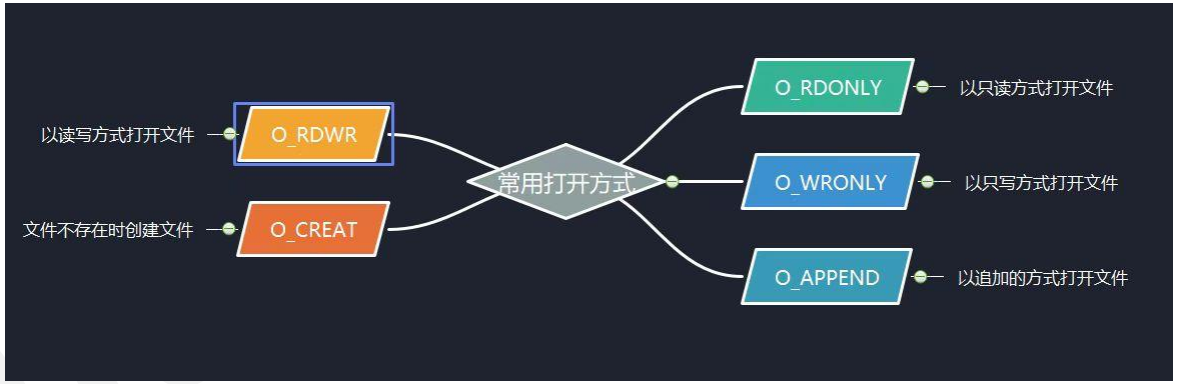
调用成功返回0，出错返回-1，并设置errno。

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。

关闭一个文件的同时，也释放该进程加在该文件上的所有记录锁。

实例

- 创建文件
  - 测试flage特性
  - 文件描述符



文件I/O – read()

调用read()函数可以从一个已打开的可读文件中读取数据。

```
#include <unistd.h>

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

读操作从文件的当前位移量处开始，在成功返回之前，该位移量增加实际读取的字节数。

fd:

文件描述符

buf：

参数需要由调用者来分配内存，并在使用后，由调用者释放分配的内存。

Count

表示缓冲区大小,一次最多读取count个数据。

read()调用成功返回读取的字节数，

如果返回0，表示到达文件末尾，

如果返回-1，表示出错，通过errno设置错误码。

文件I/O – write()

调用write()函数可以向一个已打开的可写文件中写入数据。

```
#include <unistd.h>

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

对于普通文件，写操作从文件的当前位移量处开始，如果在打开文件时，指定了O_APPEND参数，则每次写操作前，将文件位移量设置在文件的当前结尾处，在一次成功的写操作后，该文件的位移量增加实际写的字节数。

write()调用成功返回已写的字节数，失败返回-1，并设置errno。

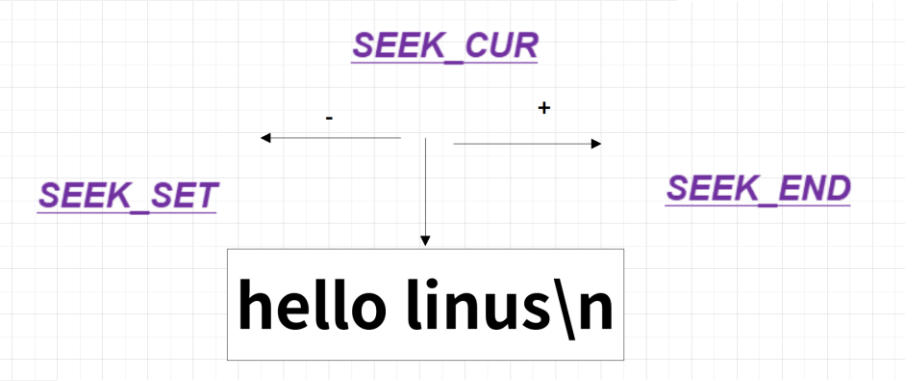
write()的返回值通常与count不同，因此需要循环将全部待写的数据全部写入文件。

write()出错的常见原因：磁盘已满或者超过了一个给定进程的文件长度限制。

实例1:读写

文件I/O – lseek()

调用 lseek()函数可以显示的定位一个已打开的文件。




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

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

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	

实例

- 测试lseek

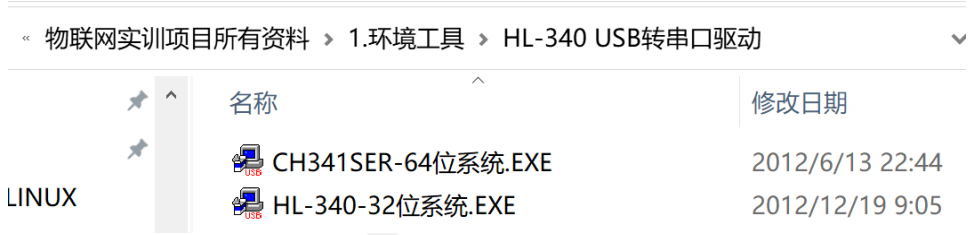
03

串口操作

串口在windows和linux下

- Windows下
  - COM3 …
- Linux下
  - ttyUSB0…
  - ttyS0…

需要驱动支持

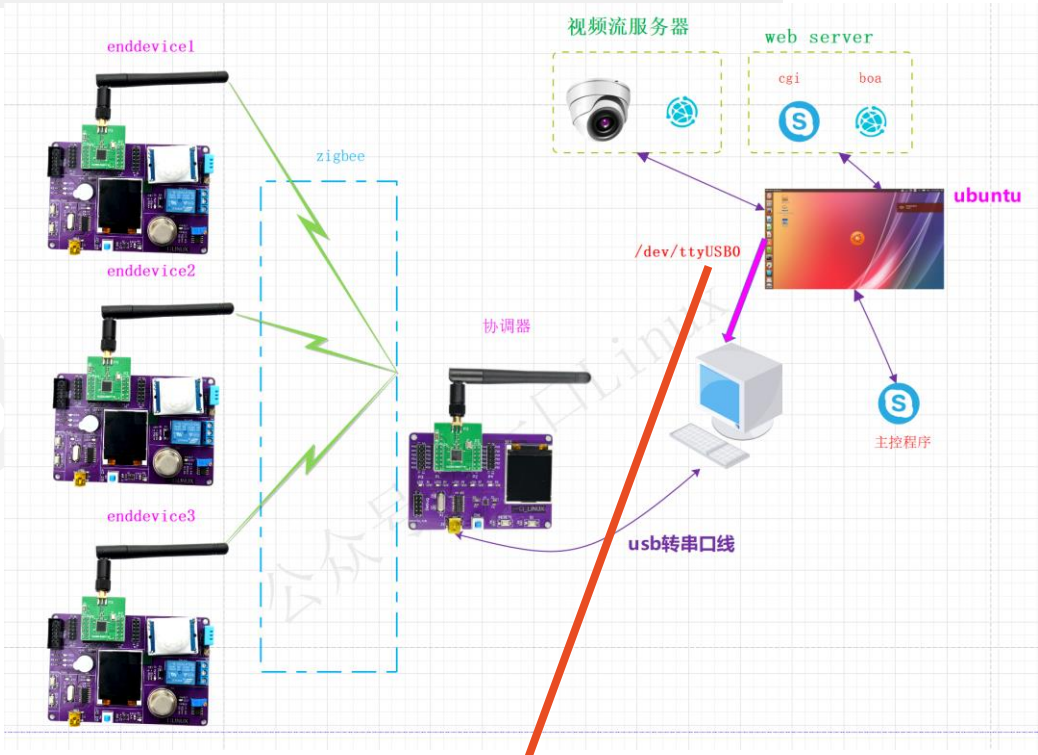


H:\物联网实训项目所有资料\3.课程代码讲解实例\3.linux系统编程\ch341SER_LINUX\ch34x.c

Linux下

1.字符设备文件节点

2.数据的读写类似于数据流




```
peng@ubuntu:~/work$ ls /dev/ttyUSB0 -l
crw-rw---- 1 root dialout 188, 0 Apr 24 06:48 /dev/ttyUSB0
```

文件类型
C：字符设备

访问权限

设备号

串口设备文件名

关注公众号：一口Linux

串口操作结构体

- /usr/include/asm-generic/termbits.h

```
struct termios
{
    unsigned short c_iflag; /* 输入模式标志*/
    unsigned short c_oflag; /* 输出模式标志*/
    unsigned short c_cflag; /* 控制模式标志*/
    unsigned short c_lflag; /*区域模式标志或本地模式标志或局部模式*/
    unsigned char c_line; /*行控制line discipline */
    unsigned char c_cc[NCC]; /* 控制字符特性*/
};
```

- c_cc：特殊控制字元可提供使用者设定一些特殊的功能

- c_iflag：标志常量：Input mode (输入模式)
  - INPCK： 启用输入奇偶检测。
  - ISTRIP： 去掉第八位
- c_cflag：标志常量：Control mode (控制模式)
  - CLOCAL： 忽略 modem 控制线
  - CREAD： 打开接受者
  - CSIZE字符长度掩码（传送或接收字元时用的位数）。 取值为CS5（传送或接收字元时用5bits）, CS6, CS7, 或 CS8
  - PARENB： 允许输出产生奇偶信息以及输入的奇偶校验（启用同位产生与侦测）
  - PARODD： 输入和输出是奇校验（使用奇同位而非偶同位）
  - CSTOPB： 设置两个停止位，而不是一个

c_cc[VTIME、VMIN]

- c_cc：特殊控制字元可提供使用者设定一些特殊的功能
- VTIME：
  - VTIME定义等待的时间，单位是百毫秒
- VMIN:
  - VMIN :定义了要求等待的最小字节数，这个字节数可能是0

- 举例：
  - MIN = 0 , TIME =0
    - 有READ立即回传否则传回 0 ,不读取任何字元
  - MIN = 0 , TIME >0
    - READ 传回读到的字元,或在十分之一秒后传回TIME若来不及读到任何字元,则传回0
  - MIN > 0 , TIME =0
    - READ 会等待,直到MIN字元可读
  - MIN > 0 , TIME > 0
    - 每一格字元之间计时器即会被启动READ 会在读到MIN字元,传回值或TIME的字元计时(1/10秒)超过时将值 传回

open未设置O_NONBLOCK或O_NDELAY的情况下

串口操作函数

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

int tcgetattr(int fd, struct termios *termios_p);

int tcsetattr(int fd, int optional_actions,
              const struct termios *termios_p);
```

取得终端介质（fd）初始值

设置与终端相关的参数
TCSANOW：改变立即发生

```
int cfsetispeed(struct termios *termios_p, speed_t speed);

int cfsetospeed(struct termios *termios_p, speed_t speed);
```

设置 termios 结构中存储的输入/出波特率为 speed

实例1:解析串口发送的环境数据

- 开发板烧录传感网完整代码
  - 2.项目完整源码\cc2530
- Ubuntu运行以下代码
  - 3.课程代码讲解实例\3.linux系统编程\3.uart\uart_r.c uart_w.c

```
ste
tem:30
hum:56
ep_no:1
hongwai:0
lux:109
gas:4
```

newtio.c_cc[VMIN] = 24 区别?

04

文件I/O进阶、错误码、库与系统调用

文件描述符

1. 文件描述符【举例】
   ① 顺序分配的非负整数
   ② 内核用以标识一个特定进程正在访问的文件
   ③ 其他资源(socket、pipe等)的访问标识
2. 标准输入、标准输出和标准出错【举例】
   ① 由shell默认打开，分别为0/1/2

文件描述符	用途	POSIX名称	stdio流	操作行为
0	标准输入	STDIN_FILENO	stdin	默认是键盘文件
1	标准输出	STDOUT_FILENO	stdout	默认是屏幕文件
2	标准错误	STDERR_FILENO	stderr	在shell中默认显示在屏幕文件上

```
lrwxrwxrwx  1 root root          15 May 29 07:28 stderr -> /proc/self/fd/2
lrwxrwxrwx  1 root root          15 May 29 07:28 stdin -> /proc/self/fd/0
lrwxrwxrwx  1 root root          15 May 29 07:28 stdout -> /proc/self/fd/1
```

fd其他知识点

1. 查看所有进程允许打开的最大 fd 数量

```
root@ubuntu:/sys/class# cat /proc/sys/fs/file-max
100984
```

2. 查看所有进程已经打开的 fd 数量以及允许的最大数量

```
root@ubuntu:/sys/class# cat /proc/sys/fs/file-nr
6368    0       100984
```

3. 查看单个进程允许打开的最大 fd 数量.

```
root@ubuntu:/sys/class# ulimit -n
1024
```

4. 查看某个文件被哪些进程打开? 【举例】

   sudo lsof *filename*

5. 查看某个进程打开了哪些文件?

   ls –l /proc/{*PID*}/fd 可以查看某个进程打开了哪些文件

```
root@ubuntu:/home/peng/driver/2/cdev# ls -l /proc/7298/fd
总用量 0
lrwx------ 1 root root 64 Aug 17 01:57 0 -> /dev/pts/0
lrwx------ 1 root root 64 Aug 17 01:57 1 -> /dev/pts/0
lrwx------ 1 root root 64 Aug 17 01:57 2 -> /dev/pts/0
lrwx------ 1 root root 64 Aug 17 01:57 3 -> /home/peng/driver/2/cdev/test
```

IO

1. **不用缓存的I/O** **【举例】**
   - ① **通过文件描述符进行访问**
   - ② open()/read()/write()/lseek()/close()…
2. **标准I/O**
   - ① **通过FILE*进行访问**
   - ② printf()/fprintf()/fopen()/fread()/fwrite()/fseek()/fclose()…
     posix

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。
- 通常，读/写操作都从**当前文件位移量处**开始，在读/写调用成功后，使位移量增加所读或者所写的字节数。
- lseek()调用成功为新的文件位移量，失败返回-1，并设置errno。
- **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**。

错误码

```
/usr/include/asm-generic/errno.h
/usr/include/asm-generic/errno-base.h
```

1. 全局错误码errno
   ① 在errno.h中定义，全局可见
   ② 错误值定义为“EXXX”形式，如EACCES
2. 处理规则
   ① 如果没有出错，则errno值不会被一个例程清除，即只有出错时，才需要检查errno值
   ② 任何函数都不会将errno值设置为0，errno.h中定义了所有常数都不为0
3. 错误信息输出
   ① strerror() - 映射errno对应的错误信息
   ② perror() – 输出用户信息及errno对应的错误信息

系统调用与库

1. 系统调用

① 用户空间进程访问内核的接口
② 把用户从底层的硬件编程中解放出来
③ 极大的提高了系统的安全性
④ 使用户程序具有可移植性

2. 库函数

① 库函数为了实现某个功能而封装起来的API集合
② 提供统一的编程接口，更加便于应用程序的移植

系统调用与库函数的区别

不涉及系统调用的库函数:
abs()、strlen()、toupper

涉及系统调用的库函数:
fopen()、fread()、fwrite()、pthread_creat()