目录

Linux 下串口编程入门教程

转贴自塞迪网
http://tech.ccidnet.com/pub/article/c302_a87895_p1.html

简介：

Linux 操作系统从一开始就对串行口提供了很好的支持，本文就 Linux 下的串行口通讯编程进行简单的介绍。

串口简介

串行口是计算机一种常用的接口，具有连接线少，通讯简单，得到广泛的使用。常用的串口是 RS-232-C 接口（又称 EIA RS-232-C）它是在 1970 年由美国电子工业协会（EIA）联合贝尔系统、 调制解调器厂家及计算机终端生产厂家共同制定的用于串行通讯的标准。它的全名是"数据终端设备（DTE）和数据通讯设备（DCE）之间串行二进制数据交换接口技术标准"该标准规定采用一个 25 个脚的 DB25 连接器，对连接器的每个引脚的信号内容加以规定，还对各种信号的电平加以规定。传输距离在码元畸变小于 4% 的情况下，传输电缆长度应为 50 英尺。

Linux 操作系统从一开始就对串行口提供了很好的支持，本文就 Linux 下的串行口通讯编程进行简单的介绍，如果要非常深入了解，建议看看本文所参考的 《Serial Programming Guide for POSIX Operating Systems》

串口操作

串口操作需要的头文件

```
#include <stdio.h> /*标准输入输出定义*/
#include <stdlib.h> /*标准函数库定义*/
#include <unistd.h> /*Unix 标准函数定义*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h> /*文件控制定义*/
#include <termios.h> /*PPSIX 终端控制定义*/
#include <errno.h> /*错误号定义*/
```

打开串口

在 Linux 下串口文件是位于 /dev 下的。串口一 为 /dev/ttyS0，串口二 为 /dev/ttyS1。打开串口是通过使用标准的文件打开函数操作：

```
int fd;
/*以读写方式打开串口*/
fd = open( "/dev/ttyS0", O_RDWR);
if (-1 == fd){
/* 不能打开串口一*/
perror(" 提示错误！ ");
}
```

设置串口

最基本的设置串口包括波特率设置，效验位和停止位设置。串口的设置主要是设置 struct termios 结构体的各成员值。

```
struct termio
{ 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; /* local mode flags */
unsigned char c_line; /* line discipline */
unsigned char c_cc[NCC]; /* control characters */
};
```

设置这个结构体很复杂，我这里就只说说常见的一些设置：

波特率设置 下面是修改波特率的代码：

```
struct termios Opt;
tcgetattr(fd, &Opt);
cfsetispeed(&Opt,B19200); /*设置为 19200Bps*/
cfsetospeed(&Opt,B19200);
tcsetattr(fd,TCANOW,&Opt);
```

设置波特率的例子函数：

```
/**
*@brief 设置串口通信速率
*@param fd 类型 int 打开串口的文件句柄
*@param speed 类型 int 串口速度
*@return void
*/
int speed_arr[] = { B38400, B19200, B9600, B4800, B2400, B1200, B300,
B38400, B19200, B9600, B4800, B2400, B1200, B300, };
int name_arr[] = {38400, 19200, 9600, 4800, 2400, 1200, 300, 38400,
19200, 9600, 4800, 2400, 1200, 300, };
void set_speed(int fd, int speed){
int i;
int status;
struct termios Opt;
tcgetattr(fd, &Opt);
for ( i= 0; i < sizeof(speed_arr) / sizeof(int); i++) {
if (speed == name_arr[i]) {
tcflush(fd, TCIOFLUSH);
cfsetispeed(&Opt, speed_arr[i]);
cfsetospeed(&Opt, speed_arr[i]);
status = tcsetattr(fd1, TCSANOW, &Opt);
if (status != 0) {
perror("tcsetattr fd1");
```

```
return;
}
tcflush(fd,TCIOFLUSH);
}
}
}
```

设置效验的函数：

```
/**
*@brief 设置串口数据位，停止位和效验位
*@param fd 类型 int 打开的串口文件句柄
*@param databits 类型 int 数据位 取值 为 7 或者 8
*@param stopbits 类型 int 停止位 取值为 1 或者 2
*@param parity 类型 int 效验类型 取值为 N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
{
struct termios options;
if ( tcgetattr( fd,&options) != 0) {
perror("SetupSerial 1");
return(FALSE);
}
options.c_cflag &= ~CSIZE;
switch (databits) /*设置数据位数*/
{
case 7:
options.c_cflag |= CS7;
break;
case 8:
options.c_cflag |= CS8;
break;
default:
fprintf(stderr,"Unsupported data sizen"); return (FALSE);
}
switch (parity)
{
case 'n':
case 'N':
options.c_cflag &= ~PARENB; /* Clear parity enable */
options.c_iflag &= ~INPCK; /* Enable parity checking */
break;
```

```
case 'o':
case 'O':
options.c_cflag |= (PARODD | PARENB); /*  设置为奇效验*/
options.c_iflag |= INPCK; /* Disnable parity checking */
break;
case 'e':
case 'E':
options.c_cflag |= PARENB; /* Enable parity */
options.c_cflag &= ~PARODD; /*  转换为偶效验*/
options.c_iflag |= INPCK; /* Disnable parity checking */
break;
case 'S':
case 's': /*as no parity*/
options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSTOPB;break;
default:
fprintf(stderr,"Unsupported parityn");
return (FALSE);
}
/*  设置停止位*/
switch (stopbits)
{
case 1:
options.c_cflag &= ~CSTOPB;
break;
case 2:
options.c_cflag |= CSTOPB;
break;
default:
fprintf(stderr,"Unsupported stop bitsn");
return (FALSE);
}
/* Set input parity option */
if (parity != 'n')
options.c_iflag |= INPCK;
tcflush(fd,TCIFLUSH);
options.c_cc[VTIME] = 150; /*  设置超时 15 seconds*/
options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
if (tcsetattr(fd,TCSANOW,&options) != 0)
{
perror("SetupSerial 3");
return (FALSE);
}
return (TRUE);
```

}

需要注意的是: 如果不是开发终端之类的，只是串口传输数据，而不需要串口来处理，那么使用原始模式(Raw Mode)方式来通讯，设置方式如下：

options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); /*Input*/
options.c_oflag &= ~OPOST; /*Output*/

读写串口

设置好串口之后，读写串口就很容易了，把串口当作文件读写就是。

·发送数据

char buffer[1024];int Length;int nByte;nByte = write(fd, buffer ,Length)

·读取串口数据

使用文件操作 read 函数读取，如果设置为原始模式(Raw Mode)传输数据，那么 read 函数返回的字符数是实际串口收到的字符数。可以使用操作文件的函数来实现异步读取，如 fcntl，或者 select 等来操作。

char buff[1024];int Len;int readByte = read(fd,buff,Len);

关闭串口

关闭串口就是关闭文件。

close(fd);

例子

下面是一个简单的读取串口数据的例子，使用了上面定义的一些函数和头文件

```
/*****************************************************************************
代码说明：使用串口二测试的，发送的数据是字符，
但是没有发送字符串结束符号，所以接收到后，后面加上了结束符号。
我测试使用的是单片机发送数据到第二个串口，测试通过。
*****************************************************************************/
#define FALSE -1
#define TRUE 0
/*****************************************************************************/
int OpenDev(char *Dev)
{
int fd = open( Dev, O_RDWR );
//| O_NOCTTY | O_NDELAY
if (-1 == fd)
{
perror("Can't Open Serial Port");
return -1;
}
else
return fd;
}
int main(int argc, char **argv){
int fd;
int nread;
char buff[512];
char *dev = "/dev/ttyS1"; //串口二
fd = OpenDev(dev);
set_speed(fd,19200);
if (set_Parity(fd,8,1,'N') == FALSE) {
printf("Set Parity Errorn");
exit (0);
}
while (1) //循环读取数据
{
while((nread = read(fd, buff, 512))>0)
{
printf("nLen %dn",nread);
buff[nread+1] = '';
printf( "n%s", buff);
}
}
//close(fd);
// exit (0);
}
```

Linux 操作系统下的串口通信学习笔记

一、什么是串口通信

串口通信是指计算机主机与外设之间以及主机系统与主机系统之间数据的串行传送。使用串口通信时，发送和接收到的每一个字符实际上都是一次一位的传送的，每一位为 1 或者为 0。

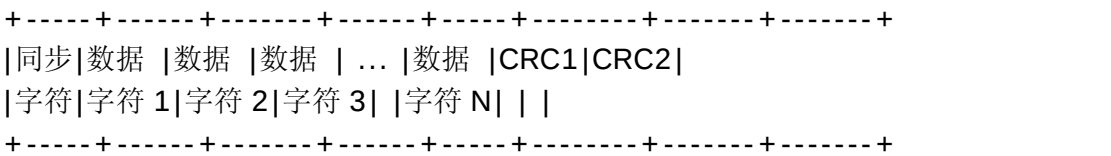
二、串口通信的分类

串口通信可以分为同步通信和异步通信两类。同步通信是按照软件识别同步字符来实现数据的发送和接收，异步通信是一种利用字符的再同步技术的通信方式。

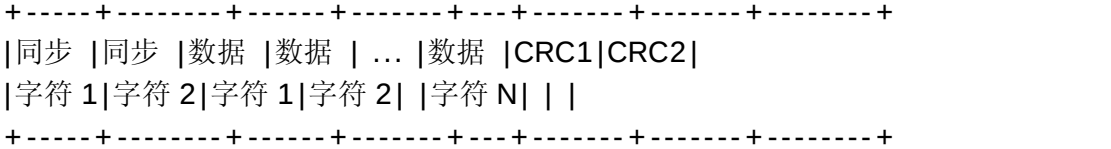
2.1 同步通信

同步通信是一种连续串行传送数据的通信方式，一次通信只传送一帧信息。这里的信息帧与异步通信中的字符帧不同，通常含有若干个数据字符。如图：

单同步字符帧结构



双同步字符帧结构



它们均由同步字符、数据字符和校验字符（CRC）组成。其中同步字符位于帧开头，用于确认数据字符的开始。数据字符在同步字符之后，个数没有限制，由所需传输的数据块长度来决定；校验字符有 1 到 2 个，用于接收端对接收到的字符序列进行正确性的校验。

同步通信的缺点是要求发送时钟和接收时钟保持严格的同步。

2.2 异步通信

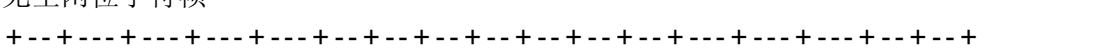
异步通信中，数据通常以字符或者字节为单位组成字符帧传送。字符帧由发送端逐帧发送，通过传输线被接收设备逐帧接收。发送端和接收端可以由各自的时钟来控制数据的发送和接收，这两个时钟源彼此独立，互不同步。

接收端检测到传输线上发送过来的低电平逻辑"0"（即字符帧起始位）时，确定发送端已开始发送数据，每当接收端收到字符帧中的停止位时，就知道一帧字符已经发送完毕。

在异步通行中有两个比较重要的指标：字符帧格式和波特率。

(1)字符帧，由起始位、数据位、奇偶校验位和停止位组成。如图：

无空闲位字符帧



|D7|0/1| 1 | 0 |D0|D1|D2|D3|D4|D5|D6|D7|0/1| 1 | 0 |D0|D1|
+--+---+---+---+--+--+--+--+--+--+--+---+---+---+--+--+
奇偶 停 起 奇偶 停 起
校验 止 始 校验 止 始
位 位 位 位
有空闲位字符帧
+---+---+--+--+--+--+--+--+--+--+---+---+---+---+---+---+--+
| 1 | 0 |D0|D1|D2|D3|D4|D5|D6|D7|0/1| 1 | 1 | 1 | 1 | 0 |D0|
+---+---+--+--+--+--+--+--+--+--+---+---+---+---+---+---+--+
空 起 奇偶 停 空 闲 位 起
闲 始 校验 止 始
位 位 位 位
1.起始位：位于字符帧开头，占 1 位，始终为逻辑 0 电平，用于向接收设备表示发送端开始发送一帧信息。
2.数据位：紧跟在起始位之后，可以设置为 5 位、6 位、7 位、8 位，低位在前高位在后。
3.奇偶校验位：位于数据位之后，仅占一位，用于表示串行通信中采用奇校验还是偶校验。
(2)波特率，波特率是每秒钟传送二进制数码的位数，单位是 b/s。
异步通信的优点是不需要传送同步脉冲，字符帧长度也不受到限制。缺点是字符帧中因为包含了起始位和停止位，因此降低了有效数据的传输速率。

三、什么是 RS-232

RS-232-C 接口（又称 EIA RS-232-C）它是在 1970 年由美国电子工业协会（EISB2.0、网卡接口、Modem 接口、VGA 接口、扩展坞、IEEE 1394 以及六合一读卡器，常用的端口一应俱全。
HP Ze2022AP

Ze2022AP 外观上采用了惠普经典的 Pavilion DV1000 的模具，模具成熟，外观无可挑剔。银白色的主色调，再经过磨砂处理，视觉效果和触感都相当理想。Ze2202AP 外形方方正正，棱角分明，同时惠普还颇费心思地在前后采用了倾斜的切割，整个外形更显精细动感。
Ze2202AP 的端口主要分布在左右两端，右侧有 2 个 USB 接口、1 个 1394 接口，SD/MS/MMC 多功能存储卡读取插槽，还有 COMBO 光驱，以及 S 端口。左侧从后到前分布电源接口、VGA 输出接口、基座扩展接口、 RJ11/RJ45 网络接口，1 个 USB 接口，以及 PCMCIA 扩展插槽。除此之外，这款机型同样配备扩展端口，扩展性无疑是相当强大的。
ThinkPad R50e 1834HC1
IBM 的 R 系列实际上是 T 系列的经济版本，有人将其形象地表示为“偷工减料版的 T”，外形上依然承袭了 IBM“小黑”一贯的酷。这款 R50e 顶盖材料采用了 ABS 工程塑料，但借助不错的顶盖框架，强度还是不错的。

R150e 的端口相对齐全，不仅有 ThinkPad 经典的指定杆，接口上还包括 RJ45/11 网络接口、两个 USB 接口、PC 卡插槽、VGA 接口、S 视频端口，但没有 IEEE1394 接口，对经常使用数码产品的消费者无疑是个坏消息。

五、全双工与半双工

1.全双工，表示机器可以同时发送数据也可以接收数据，有两个独立的数据通道（一个用于发送，一个用于接收）
2.半双工，表示机器不能在发送数据的同时也接收数据。

六、流量控制

1.使用软件方法
使用特殊的字符来标记数据流的开始和结束，比如 XON,DC1,八进制 021 来标志开始，用 X0FF,DC3,八进制 023 来标志结束。
2.使用硬件方法
使用 RS232 的 CTS 和 RTS 信号来代替特殊字符控制。当接收方准备接收更多数据时，设置 CTS 为 0,反之设置成 1。对应的发送端准备发送数据时，设置 RTS 为 0。

七、串口的访问

串口设备在 LINUX 下与所有设备一样都是通过设备文件来进行访问。

7.1 打开串口

LINUX 系统下串口设备是通过 open 函数来打开的，不过需要注意的是，一般用户是没有权限访问设备文件的，需要将打开的串口设备的访问权限设置成一般用户可以访问的权限。
open 函数
头文件
#include
#include
#include
函数原型
int open(const char *pathname, int oflag, .../*, mode_t mode*/);
参数
const char *pathname - 要打开文件的文件名称，例如/dev/ttyS0

int oflag - 文件打开方式，可用标志如下：
O_RDONLY 以只读方式打开文件
O_WRONLY 以只写方式打开文件
O_RDWR 以读写方式打开文件
O_APPEND 写入数据时添加到文件末尾
O_CREATE 如果文件不存在则产生该文件，使用该标志需要设置访问权限位 mode_t
O_EXCL 指定该标志，并且指定了 O_CREATE 标志，如果打开的文件存在则会产生一个错误
O_TRUNC 如果文件存在并且成功以写或者只写方式打开，则清除文件所有内容，使得文件长度变为 0
O_NOCTTY 如果打开的是一个终端设备，这个程序不会成为对应这个端口的控制终端，如果没有该标志，任何一个输入，例如键盘中止信号等，都将影响进程。
O_NONBLOCK 该标志与早期使用的 O_NDELAY 标志作用差不多。程序不关心 DCD 信号线的状态，如果指定该标志，进程将一直在休眠状态，直到 DCD 信号线为 0。
O_SYNC 对 I/O 进行写等待
返回值
成功返回文件描述符，如果失败返回-1
例如：以可读写方式打开/dev/ttyS0 设备

```
int fd; /* 文件描述符 */
fd = open("/dev/ttyS0", O_RDWR | 0_NOCTTY | O_NONBLOCK);
```

7.2 关闭串口

Linux 系统下通过 close 函数来关闭串口设备
close 函数
头文件
#include
函数原型

```
int close(int filedes);
```

参数
int filedes - 文件描述符
返回值
成功返回 0，否则返回-1
例如：关闭打开的串口设备 fd

```
int ret; /* 返回标志，用于判断是否正常关闭设备 */
ret = close(fd);
```

7.3 写串口

写串口是通过 write 函数来完成的
write 函数
头文件

#include
函数原型
ssize_t write(int filedes, const void *buff, size_t nbytes);
参数
int filedes - 文件描述符
const void *buff - 存储写入数据的数据缓冲区
size_t nbytes - 写入数据字节数
返回值
ssize_t - 返回写入数据的字节数,该值通常等于 nbytes，如果写入失败返回-1
例如：向终端设备发送初始化命令

```
int n = 0; /* 写入字节数 */
n = write(fd, "ATZ\r", 4);
if(n == -1)
{
fprintf(stderr, "Wirte ATZ command error.\n");
}
```

7.4 读串口

读串口是通过 read 函数来完成的
read 函数
头文件
#include
函数原型
ssize_t read(int filedes, void *buff, size_t nbytes);
参数
int filedes - 文件描述符
void *buff - 存储读取数据的数据缓冲区
size_t nbytes - 需要读取的字节数
返回值
ssize_t - 成功读取返回读取的字节数，否则返回-1
注意，在对串口进行读取操作的时候，如果是使用的 RAW 模式，每个 read 系统调用将返回当前串行输入缓冲区中存在的字节数。如果没有数据，将会一致阻塞到有字符达到或者间隔时钟到期，或者发生错误。如果想使 read 函数在没有数据的时候立即返回则可以使用 fcntl 函数来设置文件访问属性。例如：
fcntl(fd, F_SETFL, FNDELAY);
这样设置后，当没有可读取的数据时，read 函数立即返回 0。
通过 fcntl(fd, F_SETFL, 0)可以设置回一般状态。
例如：从终端读取 5 个字节的应答数据

```
int nRead; /* 从终端读取的字节数 */
char buffer[256]; /* 接收缓冲区 */
nRead = read(fd, buffer, 5);
if(nRead == -1)
```

```
{
fprintf(stderr, "Read answer message error.\n");
}
```

八、终端配置

8.1 POSIX 终端接口

大多数系统都支持 POSIX 终端接口，POSIX 终端通过一个 termios 结构来进行控制，该结构定义在 termios.h 文件中。

termios 结构

```
struct termios
{
tcflag_t c_iflag; /* 输入选项标志 */
tcflag_t c_oflag; /* 输出选项标志 */
tcflag_t c_cflag; /* 控制选项标志 */
tcflag_t c_lflag; /* 本地选项标志 */
cc_t c_cc[NCCS]; /* 控制特性 */
};
```

c_iflag 成员

Flag Description

GNBRK 忽略输入中的 BREAK 状态

BRKINT 如果设置了 IGNBRK，将忽略 BREAK。如果没有设置，但是设置了 BRKINT，那么 BREAK 将使得输入和输出队列被刷新，如果终端是一个前台进程组的控制终端，这个进程组中所有进程将收到 SIGINT 信号。如果既未设置 IGNBRK 也未设置 BRKINT，BREAK 将视为 NUL 同义字符，除非设置了 PARMRK，这种情况下被视为序列\377\0\0

IGNPAR 忽略桢错误和奇偶校验错误

PARMRK 如果没有设置 IGNPAR，在有奇偶校验错误或者桢错误的字符前插入\377\0。如果既没有设置 IGNPAR 也没有设置 PARMRK，将所有奇偶校验错误或者桢错误的字符视为\0。

INPCK 启用输入奇偶校验检测。

ISTRIP 去掉第八位。

INLCR 将输入的 NL 翻译为 CR。

IGNCR 忽略输入中的回车。

ICRNL 将输入中的回车翻译为新行字符（除非设置了 IGNCR）。

IUCLC （不属于 POSIX）将输入中的大写字母映射为小写字母。

IXON 启用输出的 XON/XOFF 流控制

IXANY （不属于 POSIX。1；XSI）允许任何字符来重新开始输出。

IXOFF 启用输入的 XON/XOFF 流控制

IMAXBEL （不属于 POSIX）当输入队列满时响铃。LINUX 没有实现该位，总是将其视为已设置。

c_oflag 成员

Flag Description

OPOST 启用具体实现自行定义的输出。
OLCUC （不属于 POSIX）将输出中的小写字母映射为大写字母。
ONLCR （XSI）将输出中的新行符映射为回车-换行
OCRNL 将输出中的回车映射为新行符。
ONOCR 不在第 0 列输出回车。
ONLRET 不输出回车。
OFILL 发送填充字符作为延时。
OFDEL （不属于 POSIX）填充字符是 ASCII DEL（0177）。如果不设置填充字符则是 ASCII NUL。
NLDLY 新行延时掩码。取值为 NL0 和 NL1。
CRDLY 回车延时掩码。取值为 CR0,CR1,CR2 或 CR3。
TABDLY 水平跳格延时掩码。取值为 TAB0,TAB1,TAB2,TAB3（或 XTABS）。取值为 TAB3,即 XTABS，将扩展跳格为空格（每个跳格符填充 8 个空格）。
BSDLY 回车延时掩码。取值为 BS0 或 BS1.（从来没有被实现）
VTDLY 竖直跳格掩码。取值为 VT0 或 VT1。
FFDLY 进表延时掩码。取值为 FF0 或者 FF1。
c_cflag 成员
Flag Description
CBAUD （不属于 POSIX）波特率掩码（4+1 位）。
CBAUDEX （不属于 POSIX）扩展的波特率掩码（1 位），包含在 CBAUD 中。
CSIZE 字符长度掩码。取值为 CS5,CS6,CS7 或 CS8。
CSTOPB 设置两个停止位。
CREAD 打开接受者。
PARENB 允许输出产生奇偶信息以及输入的奇偶校验。
PARODD 输入和输出是奇校验
HUPCL 在最后一个进程关闭设备后，降低 MODEM 控制线（挂断）。
CLOCAL 忽略 MODEM 控制线。
LOBLK （不属于 POSIX）从非当前 SHELL 层阻塞输出（用于 sh1）。
CIBAUD（不属于 POSIX）输入速度的掩码。CIBAUD 各位的值与 CBAUD 各位相同，左移了 IBSHIFT 位。
CRTSCTS （不属于 POSIX）启用 RTS/CTS（硬件）控制流。

c_lflag 成员
Flag Description
ISIG 当接收到字符 INTR，QUIT，SUSP 或 DSUSP 时，产生相应的信号。
XCASE （不属于 POSIX；LINUX 下不支持）如果同时设置了 ICANON，终端只有大写。输入被转换为小写，除了以\前缀的字符。输出时，大写字符被前缀\，小写字符被转换成大写。
ECHO 回显输入字符。
ECHOE 如果同时设置了 ICANON，字符 ERASE 擦除前一个输入字符，WERASE 擦除前一个词。
ECHOK 如果同时设置了 ICANON，字符 KILL 删除当前行。
ECHONL 如果同时设置了 ICANON，回显字符 NL，即使没有设置 ECHO。

ECHOCTL （不属于 POSIX）如果同时设置了 ECHO，除了 TAB，NL，START 和 STOP 之外的 ASCII 控制信号被回显为^x，这里 X 是比控制信号大 0x40 的 ASCII 码。例如字符 0x08(BS)被回显为^H。
ECHOPRT （不属于 POSIX）如果同时设置了 ICANON 和 IECHO，字符在删除的同时被打印。
ECHOKE （不属于 POSIX）如果同时设置了 ICANON，回显 KILL 时将删除一行中的每个字符，如同指定了 ECHOE 和 ECHORPT 一样。
DEFECHO （不属于 POSIX）只在一个进程读的时候回显。
FLUSHO （不属于 POSIX；LINUX 不支持）输出被刷新。这个标志可以通过键入字符 DISCARD 来打开和关闭。
NOFLSH 禁止产生 SIGINT，SIGQUIT 和 SIGSUSP 信号时刷新输入和输出队列。
TOSTOP 向试图写控制终端的后台进程组发送 SIGTTOU 信号。
PENDIN （不属于 POSIX；LINUX 不支持）在读入一个字符时，输入队列中的所有字符被重新输出。（bash 用他来处理 typeahead）。
IEXTEN 启用实现自定义的输入处理。这个标志必须与 ICANON 同时使用，才能解释特殊字符 EOL2,LNEXT，REPRINT 和 WERASE，IUCLC 标志才有效。
c_cc 数组成员
Flag Description
VINTR （003,ETX，Ctrl-C,or also 0177, DEL, rubout）中断字符。发送 SIGINT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VQUIT （034,FS，Ctrl-\）退出字符。发出 SIGQUIT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VERASE (0177, DEL, rubout, or 010, BS, Ctrl-H, or also #) 删除字符。删除上一个还没有删掉的字符，但不删除上一个 EOF 或行首。当设置 ICANON 时可被识别，不再作为输入传递。
VKILL (025, NAK, Ctrl-U, or Ctrl-X, or also @) 终止字符。删除自上一个 EOF 或行首以来的输入。当设置 ICANON 时可被识别，不再作为输入传递。
VEOF (004, EOT, Ctrl-D) 文件尾字符。更精确地说，这个字符使得 tty 缓冲中的内容被送到等待输入的用户程序中，而不必等到 EOL。如果它是一行的第一个字符，那么用户程序的 read() 将返回 0，指示读到了 EOF。当设置 ICANON 时可被识别，不再作为输入传递。
VMIN 非 canonical 模式读的最小字符数。 VEOL (0, NUL) 附加的行尾字符。当设置 ICANON 时可被识别。 VTIME 非 canonical 模式读时的延时，以十分之一秒为单位。 VEOL2 (not in POSIX; 0, NUL) 另一个行尾字符。当设置 ICANON 时可被识别。
VEOL (0, NUL) 附加的行尾字符。当设置 ICANON 时可被识别。
VTIME 非 canonical 模式读时的延时，以十分之一秒为单位。
VEOL2 (not in POSIX; 0, NUL) 另一个行尾字符。当设置 ICANON 时可被识别。
VSWTCH (not in POSIX; not supported under Linux; 0, NUL) 开关字符。(只为 shl 所用。)
VSTART (021, DC1, Ctrl-Q) 开始字符。重新开始被 Stop 字符中止的输出。当设置 IXON 时可被识别，不再作为输入传递。
VSTOP (023, DC3, Ctrl-S) 停止字符。停止输出，直到键入 Start 字符。当设置 IXON 时可被识别，不再作为输入传递。

VSUSP (032, SUB, Ctrl-Z) 挂起字符。发送 SIGTSTP 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VDSUSP (not in POSIX; not supported under Linux; 031, EM, Ctrl-Y) 延时挂起信号。当用户程序读到这个字符时，发送 SIGTSTP 信号。当设置 IEXTEN 和 ISIG，并且系统支持作业管理时可被识别，不再作为输入传递。
VLNEXT (not in POSIX; 026, SYN, Ctrl-V) 字面上的下一个。引用下一个输入字符，取消它的任何特殊含义。当设置 IEXTEN 时可被识别，不再作为输入传递。
VWERASE (not in POSIX; 027, ETB, Ctrl-W) 删除词。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VREPRINT (not in POSIX; 022, DC2, Ctrl-R) 重新输出未读的字符。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VDISCARD (not in POSIX; not supported under Linux; 017, SI, Ctrl-O) 开关：开始/结束丢弃未完成的输出。当设置 IEXTEN 时可被识别，不再作为输入传递。
VSTATUS (not in POSIX; not supported under Linux; status request: 024, DC4, Ctrl-T).

8.2 设置波特率
对于波特率的设置通常使用 cfsetospeed 和 cfsetispeed 函数来完成。获取波特率信息是通过 cfgetispeed 和 cfgetospeed 函数来完成的。
cfsetospeed 函数
头文件:
#include
函数原型:
int cfsetospeed(struct termios *termptr, speed_t speed);
参数：
struct termios *termptr - 指向 termios 结构的指针
speed_t speed - 需要设置的输出波特率
返回值：
如果成功返回 0,否则返回-1
cfsetispeed 函数
头文件：
#include
函数原型:
int cfsetispeed(struct termios *termptr, speed_t speed);
参数:
struct termios *termptr - 指向 termios 结构的指针
speed_t speed - 需要设置的输入波特率
返回值：
如果成功返回 0,否则返回-1
cfgetospeed 函数
头文件：
#include
函数原型：
speed_t cfgetospeed(const struct termios *termptr);

参数:
const struct termios - 指向 termios 结构的指针
返回值：
返回输出波特率
cfgetispeed 函数
头文件：
#include
函数原型:
speed_t cfgetispeed(const struct termios *termptr);
参数:
const struct termios *termptr - 指向 termios 结构的指针
返回值：
返回输入波特率
波特率常量：
CBAUD 掩码
B0 0 波特
B50 50 波特
B75 75 波特
B110 100 波特
B134 134 波特
B150 150 波特
B200 200 波特
B300 300 波特
B600 600 波特
B1200 1200 波特
B1800 1800 波特
B2400 2400 波特
B9600 9600 波特
B19200 19200 波特
B38400 38400 波特
B57600 57600 波特
B115200 115200 波特

8.3 设置字符大小
设置字符的大小通过设置 c_cflag 标志位来实现的。
例如：
option.c_cflag &= ~CSIZE;
option.c_cflag |= CS7;

8.4 设置奇偶校验
对于奇偶校验是需要手工设置的，常用的设置方式如下：
No parity (8N1):
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB

options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
Even parity (7E1):
options.c_cflag |= PARENB
options.c_cflag &= ~PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;
Odd parity (7O1):
options.c_cflag |= PARENB
options.c_cflag |= PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;
Space parity is setup the same as no parity (7S1):
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

8.5 获取和设置终端属性
设置和获取终端控制属性是通过 tcgetattr 和 tcsetattr 两个函数来完成的
tcgetattr 函数
头文件:
#include
函数原型:
int tcgetattr(int filedes, struct termios *termptr);
参数：
int filedes - 文件描述符
struct termiso *termptr - 指向 termios 结构的指针，
返回值:
如果成功返回 0,否则返回-1
tcsetattr 函数
头文件:
#include
函数原型：
int tcsetattr(int filedes, int opt, const struct termios *termptr);
参数：
int filedes - 文件描述符
int opt - 选项值，可以为下面三个值之一
TCSANOW - 不等数据传输完毕就改变属性
TCSADRAIN - 等待所有数据传输结束才改变属性
TCSAFLUSH - 清空输入输出缓冲区并且是设置属性
const struct termios *termptr - 指向 termios 结构的指针，

返回值：
成功返回 0,否则返回-1

九、常用设置

9.1 设置规范模式
规范模式是面向行的输入方式，输入字符被放入用于和用户交互可以编辑的缓冲区内，直接到读入回车或者换行符号时才结束。
可以通过如下方式来设置
```
option.c_lflag |= (ICANON | ECHO | ECHOE);
```

9.2 设置原始输入模式
原始输入模式是没有处理过的，当接收数据时，输入的字符在它们被接收后立即被传送，使用原始输入模式时候，一般可以选择取消 ICANON，ECHO，ECHOE 和 ISIG 选项。
例如：
```
option.c_lflag &= ~(ICANON | ECHO | ECHOE);
```

9.3 设置输入奇偶选项
当激活 c_cflag 中的奇偶校验后，应该激活输入的奇偶校验。与之相关的标志有 INPCK，IGNPAR，PARMRK 和 ISTRIP。一般是通过选择 INPCK 和 ISTRIP 激活检验和移除奇偶位。
例如：
```
option.c_iflag |= (INPCK | ISTRIP);
```

9.4 设置软件控制流
软件控制流通过 IXON，IXOFF 和 IXANY 标志来设置
例如：
```
option.c_iflag |=(IXON | IXOFF | IXANY);
```

9.5 选择预处理输出
通过 OPOST 标志来设置预处理的输出
例如：
```
option.c_oflag |= OPOST;
```

9.6 选择原始数据输出
原始数据的输出通过设置 c_oflag 的 OPOST 标志
例如:
```
option.c_oflag &= ~OPOST;
```

9.7 设置软件流控制字符
软件流控制字符是通过 c_cc 数组中的 VSTART 和 VSTOP 来设置的，一般来说，它们应该被设置城 DC1（021 八进制）和 DC3（023 八进制），分别表示 ASCII 码的 XON 和 XOFF 字符。

9.8 设置读超时

c_cc 数组中的 VMIN 指定了最少读取的字符数，如果设置为 0,那么 VTIME 就指定了读取每个字符的等待时间。VTIME 是以 1/10 秒为单位指定接收字符的超时时间的，如果 VTIME 设置为 0,而端口没有用 open 或者 fcntl 设置为 NONBLOCK，那么 read 操作将会阻塞不确定的时间。