专题6-Linux 下串口编程入门

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》

计算机串口的引脚说明

序号	信号名称	符号	流向	功能
2	发送数据	TXD	DTE→DCE	DTE发送串行数据
3	接收数据	RXD	DTE←DCE	DTE 接收串行数据
4	请求发送	RTS	DTE→DCE	DTE 请求 DCE 将线路切换到发送方式
5	允许发送	CTS	DTE←DCE	DCE 告诉 DTE 线路已接通可以发送数据
6	数据设备准备好	DSR	DTE←DCE	DCE 准备好
7	信号地			信号公共地
8	载波检测	DCD	DTE←DCE	表示 DCE 接收到远程载波
20	数据终端准备好	DTR	DTE→DCE	DTE 准备好
22	振铃指示	RI	DTE←DCE	表示 DCE 与线路接通，出现振铃

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);
    }
  }
}
```

效验位和停止位的设置：

无效验	8位	**Option.c_cflag &= ~PARENB; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag |= ~CS8;**
奇效验(Odd)	7位	Option.c_cflag |= ~PARENB; Option.c_cflag &= ~PARODD; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag |= ~CS7;

无效验	**8位**	**Option.c_cflag &= ~PARENB; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag |= ~CS8;**
偶效验(Even)	7位	Option.c_cflag &= ~PARENB; Option.c_cflag |= ~PARODD; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= ~CSIZE; Option.c_cflag |= ~CS7;
Space效验	7位	Option.c_cflag &= ~PARENB; Option.c_cflag &= ~CSTOPB; Option.c_cflag &= &~CSIZE; Option.c_cflag |= CS8;

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 size\n"); 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 parity\n");
        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 bits\n");
         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 Error\n");
        exit (0);
    }
while (1) //循环读取数据
{
    while((nread = read(fd, buff, 512))>0)
    {
        printf("\nLen %d\n",nread);
        buff[nread+1] = '\0';
        printf( "\n%s", buff);
    }
}
    //close(fd);
    // exit (0);
}
```