Linux操作系统下串口编程入门教程

　　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 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);

}

参考资料

tcflush()funciotn

<http://pubs.opengroup.org/onlinepubs/7908799/xsh/tcflush.html>

cfsetospeed() function

<http://pubs.opengroup.org/onlinepubs/7908799/xsh/cfsetospeed.html>

tcsetattr() function

<http://pubs.opengroup.org/onlinepubs/009695399/functions/tcsetattr.html>

cfsetispeed() funciton

<http://pubs.opengroup.org/onlinepubs/7908799/xsh/cfsetispeed.html>

tcgetattr() funciotns

<http://pubs.opengroup.org/onlinepubs/009695399/functions/tcgetattr.html>

Serial Programming Guide for POSIX Operating Systems