linux 下串口编程简单实例

1、 linux 中的串口设备文件存放于/dev 目录下，其中串口一，串口二对应设备名依次为“/dev/ttyS0”、“/dev/ttyS1”。在 linux 下操作串口与操作文件相同。

2、 在使用串口之前必须设置相关配置，包括：波特率、数据位、校验位、停止位等。串口设置由下面结构体实现：

```
de>struct termios{
    tcflag_t c_iflag;       /*input flags*/
    tcflag_t c_oflag;       /*output flags*/
    tcflag_t c_cflag;       /*control flags*/
    tcflag_t c_lflag;       /*local flags*/
    cc_t c_cc[NCCS];          /*control characters*/
};de>
```

该结构中 c_cflag 最为重要，可设置波特率、数据位、校验位、停止位。在设置波特率时需在数字前加上‘B’，如 B9600、B19200。使用其需通过“与”“或”操作方式。

常用的串口控制函数：

Tcgetattr 取属性(termios 结构)

Tcsetattr 设置属性(termios 结构)

cfgetispeed 得到输入速度

Cfgetospeed 得到输出速度

Cfsetispeed 设置输入速度

Cfsetospeed 设置输出速度

tcflush 刷清未决输入和/或输出

3、 串口的配置

(1) 保存原先串口配置使用 tcgetattr(fd,&oldtio)函数：

struct termios newtio,oldtio;

tcgetattr(fd,&oldtio);

(2) 激活选项有 CLOCAL 和 CREAD,用于本地连接和接收使能。

newtio.c_cflag | = CLOCAL | CREAD;

(3) 设置波特率，使用函数 cfsetispeed、 cfsetospeed

cfsetispeed(&newtio, B115200);

cfsetospeed(&newtio, B115200);

(4) 设置数据位，需使用掩码设置。

newtio.c_cflag &= ~CSIZE;

newtio.c_cflag |= CS8;

(5) 设置奇偶校验位，使用 c_cflag 和 c_iflag。

设置奇校验：

newtio.c_cflag |= PARENB;

newtio.c_cflag |= PARODD;

newtio.c_iflag |= (INPCK | ISTRIP);

设置偶校验：

newtio.c_iflag |= (INPCK | ISTRIP);

newtio.c_cflag |= PARENB;

newtio.c_cflag &= ~PARODD;

(6) 设置停止位，通过激活 c_cflag 中的 CSTOPB 实现。若停止位为 1，则清除 CSTOPB，若停止位为 2，则激活 CSTOPB。

newtio.c_cflag &= ~CSTOPB;

(7) 设置最少字符和等待时间，对于接收字符和等待时间没有特别要求时，可设为 0。

newtio.c_cc[VTIME] = 0;

newtio.c_cc[VMIN] = 0;

(8) 处理要写入的引用对象

tcflush 函数刷清（抛弃）输入缓存（终端驱动程序已接收到，但用户程序尚未读）或输出缓存（用户程序已经写，但尚未发送）。

int tcflush(int filedes, int queue)

queue 数应当是下列三个常数之一：

? TCIFLUSH 刷清输入队列。

? TCOFLUSH 刷清输出队列。

? TCIOFLUSH 刷清输入、输出队列。

如：tcflush(fd,TCIFLUSH);

(9) 激活配置。在完成配置后，需激活配置使其生效。使用 tsettattr()函数。原型：

int tcgetattr(int filedes, struct termios * termptr);

int tcsetattr(int filedes, int opt, const struct termios * termptr);

tcsetattr 的参数 opt 使我们可以指定在什么时候新的终端属性才起作用。opt 可以指定为下列常数中的一个：

? TCSANOW 更改立即发生。

? TCSADRAIN 发送了所有输出后更改才发生。若更改输出参数则应使用此选择项。

? TCSAFLUSH 发送了所有输出后更改才发生。更进一步，在更改发生时未读的所有输入数据都被删除（刷清）

使用如：tcsetattr(fd,TCSANOW,&newtio)

4、 在配置完串口的相关属性后，就可对串口进行打开，读写操作了。其使用方式与文件操作一样，区别在于串口是一个终端设备。

(1) 打开串口

fd = open("/dev/ttyS0", O_RDWR|O_NOCTTY|O_NDELAY);

Open 函数中除普通参数外，另有两个参数 O_NOCTTY 和 O_NDELAY。

O_NOCTTY: 通知 linux 系统，这个程序不会成为这个端口的控制终端。

O_NDELAY: 通知 linux 系统不关心 DCD 信号线所处的状态（端口的另一端是否激活或者停止）。

(2) 恢复串口的状态为阻塞状态，用于等待串口数据的读入。用 fcntl 函数：

fcntl（fd, F_SETFL, 0）;

(3) 接着，测试打开的文件描述府是否引用一个终端设备，以进一步确认串口是否正确打开。

isatty(STDIN_FILENO);

(4) 串口的读写与普通文件一样，使用 read,write 函数。

read(fd,buf,8);

write(fd,buf,8);

以下为一简单的程序实例：

```
de>#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <stdlib.h>

int set_opt(int fd,int nSpeed, int nBits, char nEvent, int nStop)
{
  struct termios newtio,oldtio;
  if ( tcgetattr( fd,&oldtio) != 0) {
    perror("SetupSerial 1");
    return -1;
  }
  bzero( &newtio, sizeof( newtio ) );
  newtio.c_cflag |= CLOCAL | CREAD;
  newtio.c_cflag &= ~CSIZE;
```

```
 switch( nBits )
 {
 case 7:
   newtio.c_cflag |= CS7;
   break;
 case 8:
   newtio.c_cflag |= CS8;
   break;
 }

 switch( nEvent )
 {
 case 'O':
   newtio.c_cflag |= PARENB;
   newtio.c_cflag |= PARODD;
   newtio.c_iflag |= (INPCK | ISTRIP);
   break;
 case 'E':
   newtio.c_iflag |= (INPCK | ISTRIP);
   newtio.c_cflag |= PARENB;
   newtio.c_cflag &= ~PARODD;
   break;
 case 'N':
   newtio.c_cflag &= ~PARENB;
   break;
 }

switch( nSpeed )
 {
 case 2400:
   cfsetispeed(&newtio, B2400);
   cfsetospeed(&newtio, B2400);
   break;
 case 4800:
   cfsetispeed(&newtio, B4800);
   cfsetospeed(&newtio, B4800);
   break;
 case 9600:
   cfsetispeed(&newtio, B9600);
   cfsetospeed(&newtio, B9600);
   break;
 case 115200:
   cfsetispeed(&newtio, B115200);
```

```
    cfsetospeed(&newtio, B115200);
    break;
  default:
    cfsetispeed(&newtio, B9600);
    cfsetospeed(&newtio, B9600);
    break;
  }
  if( nStop == 1 )
    newtio.c_cflag &= ~CSTOPB;
  else if ( nStop == 2 )
  newtio.c_cflag |= CSTOPB;
  newtio.c_cc[VTIME] = 0;
  newtio.c_cc[VMIN] = 0;
  tcflush(fd,TCIFLUSH);
  if((tcsetattr(fd,TCSANOW,&newtio))!=0)
  {
    perror("com set error");
    return -1;
  }
  printf("set done!\n");
  return 0;
}

int open_port(int fd,int comport)
{
  char *dev[]={"/dev/ttyS0","/dev/ttyS1","/dev/ttyS2"};
  long vdisable;
  if (comport==1)
  { fd = open( "/dev/ttyS0", O_RDWR|O_NOCTTY|O_NDELAY);
    if (-1 == fd){
      perror("Can't Open Serial Port");
      return(-1);
    }
    else
      printf("open ttyS0 .....\n");
  }
  else if(comport==2)
  { fd = open( "/dev/ttyS1", O_RDWR|O_NOCTTY|O_NDELAY);
    if (-1 == fd){
      perror("Can't Open Serial Port");
      return(-1);
    }
    else
      printf("open ttyS1 .....\n");
```

```
  }
  else if (comport==3)
  {
    fd = open( "/dev/ttyS2", O_RDWR|O_NOCTTY|O_NDELAY);
    if (-1 == fd){
      perror("Can't Open Serial Port");
      return(-1);
    }
    else
      printf("open ttyS2 .....\n");
  }
  if(fcntl(fd, F_SETFL, 0)<0)
    printf("fcntl failed!\n");
  else
    printf("fcntl=%d\n",fcntl(fd, F_SETFL,0));
  if(isatty(STDIN_FILENO)==0)
    printf("standard input is not a terminal device\n");
  else
    printf("isatty success!\n");
  printf("fd-open=%d\n",fd);
  return fd;
}
int main(void)
{
  int fd;
  int nread,i;
  char buff[]="Hello\n";
  if((fd=open_port(fd,1))<0){
    perror("open_port error");
    return;
  }
  if((i=set_opt(fd,115200,8,'N',1))<0){
    perror("set_opt error");
    return;
  }
  printf("fd=%d\n",fd);
  nread=read(fd,buff,8);
  printf("nread=%d,%s\n",nread,buff);
  close(fd);
  return;
}de>
```

linux 下的串口通讯源程序－－（测试版）

根据前面文章讲到的内容，为了说明问题，下面给出测试程序来理解 linux 下的串口操作流程，例程 receive.c 用来接收从串口发来的数据，而例程 send.c 用来发送数据到串口。二者成功建立串口连接后，串口接收端会收到串口发送端发来的字符串数据“Hello，this is a Serial Port test!”。

1. receive.c 程序清单：

```
/*********************************************************
*ilename：receive.c
* Description：Receive data from Serial_Port
* Date：
*********************************************************/
/*********************头文件定义**********************/
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include "math.h"
#define max_buffer_size    100   /*定义缓冲区最大宽度*/
/*********************************************************/
int fd，s;
int open_serial(int k)
{
if(k==0)       /*串口选择*/
{
fd = open("/dev/ttyS0",O_RDWR|O_NOCTTY);   /*读写方式打开串口*/
perror("open /dev/ttyS0");
}
```

```
else
{
fd = open("/dev/ttyS1",O_RDWR|O_NOCTTY);
perror("open /dev/ttyS1");
}
if(fd == -1)  /*打开失败*/
return -1;
else
return 0;
}
/*****************************************************************/
int main()
{
char   hd[max_buffer_size],*rbuf; /*定义接收缓冲区*/
int flag_close, retv,i,ncount="0";
struct termios opt;
int realdata="0";
/*****************************************************************/
open_serial(0);    /*打开串口 1*/
/*****************************************************************/
tcgetattr(fd,&opt);
cfmakeraw(&opt);
/*****************************************************************/
cfsetispeed(&opt,B9600); /*波特率设置为 9600bps*/
cfsetospeed(&opt,B9600);
/*****************************************************************/
tcsetattr(fd,TCSANOW,&opt);
rbuf="hd"; /*数据保存*/
printf("ready for receiving data...\n");
retv="read"(fd,rbuf,1);   /*接收数据*/
```

```
if(retv==-1)
{
perror("read"); /*读状态标志判断*/
}
/************************开始接收数据*****************************/
while(*rbuf!='\n')       /*判断数据是否接收完毕*/
{
ncount+=1;
rbuf++;
retv="read"(fd,rbuf,1);
if(retv==-1)
{
perror("read");
}
}
/*********************************************************************/
printf("The data received is:\n");   /*输出接收到的数据*/
for(i="0";i<ncount;i++)
{
printf("%c",hd[i]);
}
printf("\n");
flag_close =close(fd);
if(flag_close ＝＝－1)   /*判断是否成功关闭文件*/
printf(“Close the Device failur！ \n”);
return 0;
}
/***************************结束*********************************/
```

2.send.c 程序清单

```
/*******************************************************
```

```
* File Name：        send.c
* Description：     send data to serial_Port
* Date：
*********************************************************/
/*****************头文件定义*****************/
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#define max_buffer_size    100    /*定义缓冲区最大宽度*/
/*****************************************/
int fd;   /*定义设备文件描述符*/
int flag_close;
int open_serial(int k)
{
if(k==0)       /*串口选择*/
{
fd = open("/dev/ttyS0",O_RDWR|O_NOCTTY);   /*读写方式打开串口*/
perror("open /dev/ttyS0");
}
else
{
fd = open("/dev/ttyS1",O_RDWR|O_NOCTTY);
perror("open /dev/ttyS1");
}
if(fd == -1)   /*打开失败*/
```

```
return -1;
else
return 0;
}
/*********************************************************************/

int main(int argc, char *argv[ ] )
{
char sbuf[]={"Hello,this is a Serial_Port test!\n"};/*待发送的内容，以\n 为结束标志*/
int sfd,retv,i;
struct termios option;
int length="sizeof"(sbuf);/*发送缓冲区数据宽度*/
/*********************************************************************/
open_serial(0);    /*打开串口 1*/
/*********************************************************************/
printf("ready for sending data...\n"); /*准备开始发送数据*/
tcgetattr(fd,&option);
cfmakeraw(&option);
/*********************************************************************/
cfsetispeed(&opt,B9600); /*波特率设置为 9600bps*/
cfsetospeed(&opt,B9600);
/*********************************************************************/
tcsetattr(fd,TCSANOW,&option);
retv="write"(fd,sbuf,length);   /*接收数据*/
if(retv==-1)
{
perror("write");
}
printf("the number of char sent is %d\n",retv);
```

```
flag_close =close(fd);
if(flag_close ＝＝－1)  /*判断是否成功关闭文件*/
printf("Close the Device failur！\n");

return 0;
}
/**************************结束**********************************/
```

分别将上面的俩个程序编译之后就可以运行了，如果是在两个不同的平台上运行，比如，在开发板上运行数据发送程序 write（write.c 编译后得到），在宿主机上运行结收数据程序 read（read.c 编译得到），采用串口线将二者正确连接之后，就可以运行来看实际的效果了：

首先在宿主机端运行数据接收程序 receive：

```
[zhang@localhost]# ./receive
[zhang@localhost]#open /dev/ttyS0： Success
ready for receiving data...
The data received is：
Hello,this is a Serial_Port test!
[zhang@localhost]#
```

在接收端运行完程序之后再到发送端运行数据发送程序 send：

```
#./send
ready for sending data...
the number of char sent is 35
#
```

运行完发送程序之后就可以在接收端看到接收的数据了。也可以在一台 PC 机上来运行这两个程序，这时需要将串口线的 2、3 脚短路连接即可（自发自收），实际运行的步骤与上面相同