

使用Socket编程访问IP 串口

一. 简介

在很多情况下，用户希望自己通过Socket函数编写程序访问LIANDA。因为Socket函数几乎所有操作系统都支持，所以使用Socket访问LIANDA具有良好的跨平台性，另外使用Socket函数要比Windows下或Unix的串口访问函数简单的多的多，还有，几乎所有编写过网络程序的程序员都会使用Socket函数。之所以提供COM口和TTY口的驱动的主要目的是为了让LIANDA直接支持户以前编写的基于串口的应用程序。我们鼓励基于LIANDA开发新应用程序的用户使用Socket函数访问LIANDA。

二. 配置命令SCKCFG

```
sckcfg [N] [-m MODE] [-rip R_IP] [-rt R_PORT] [-lt L_PORT] [-c CONN] [-cs CSEQ] [-scs SCSEQ] [-d DISC] [-ds DSEQ] [-sds SDSEQ] [-k KEEPALV]
```

[N] 对应设备上串口号, 支持X-Y批量设置

-m MODE 工作模式, 取值tc|ts|u

tc - TCP 客户端模式

ts - TCP 服务器端模式

u - UDP 模式

-rip R_IP 远端主机IP地址

-rt R_PORT 远端主机TCP/UDP 端口号

-lt R_PORT 本地设备(LIANDA)TCP/UDP 端口号

-c CONN 初始化连接方式取值n | dcdon | dsron | seq | brk

n - 从不初始化连接

dcdon - 当DCD信号为高时, 初始化连接

dsron - 当DSR 信号为高时, 初始化连接

seq - 当收到 CSEQ (见下 -cs), 初始化连接

brk - 当收到BREAK 信号, 初始化连接

-cs CSEQ 指定串口接收到的初始化连接的序列, 取值范围为十六进制数“0-9”“A-F”

-scs SCSEQ 收到CSEQ 后是否发送CSEQ 给主机, 取值1|0

1 - 发送CSEQ

0 - 不发送CSEQ

-d DISC 关闭连接方式, 取值 n | dcdoff | dsroff | seq | brk

n - 从不关闭连接

dcdoff - 当DCD信号为低时, 关闭连接

dsroff - 当DSR 信号为低时, 关闭连接

seq - 当收到 DSEQ (见下 -ds), 关闭连接

brk - 当收到 BREAK 信号, 关闭连接

-ds DSEQ 指定串口接收到的关闭连接的序列, 取值范围为十六进制数“0-9”“A-F”

-sds SCSEQ 收到DSEQ 后是否发送DSEQ 给主机, 取值1|0

1 - 发送 DSEQ

0 - 不发送DSEQ

-k KEEPALV 以秒计的SOCKET保活时间, 取值范围0-10000, 0 表示无效

注意事项:

- 子选项-rip和-rt分别配置了主机端的IP地址和端口号. 当工作模式为tc或u 时, 两者合起来建立主机端的SOCKET. 当工作模式为ts 时, 两者若有配置值, 表示只能由它们建立的SOCKET 才可以访问LIANDA 的这个串口, 否则不加限制.
- 子选项-lt 配置了设备端(LIANDA)的端口号, 它和LIANDA 的IP 地址合起来建立设备端的SOCKET. 当工作模

式为tc 是无效,工作模式为ts 或u时有效.

3. 子选项-c 配置初始化连接方式. 它当工作在tc 的模式下时, 在何种条件下LIANDA 去发起连接. 其中当配置值为seq 时, 就必须配置后面的-cs 和-scs 这两个子选项, 否则这两个子选项无效.

4. 子选项-d 配置关闭连接方式. 它表示在何种条件下LIANDA 主动去关闭连

接. 其中当配置值为seq 时, 就必须配置后面的-ds 和-sds 这两个子选项, 否则这两个子选项无效.

5. -k 子选项配置的保活时间值, 是对设备(LIANDA)而言. 主机端程序的保活处理必须在程序程序中另做处理.

三. 工作模式

LIANDA 共有三种SOCKET访问模式:

1. TCP SERVER模式

在这种模式下, LIANDA 处于Listen 状态, 监听的端口号是命令SCKCFG 中子选项 -lt的配置值.

2. TCP CLIENT模式

在这种模式下, LIANDA 主动发起连接. LIANDA 所要连接的主机的IP 地址和端口号分别是命令SCKCFG中子选项 -rip和-rt的值.

3. UDP模式

在这种模式下, LIANDA 处于UDP server/client状态. LIANDA 端的SOCKET绑定(bind)在值为SCKCFG 中子选项-lt 的配置参数的端口上. 对等端的SOCKET 则绑定在IP 地址为RIP 的主机的端口RPORT上.

(其中RIP和RPORT分别为命令SCKCFG 中子选项-rip和-rt的值)

四. 协议包格式

注意事项:

1. 命令和数据使用同一个SOCKET
2. 命令以0xff开头, 数据中的0xff转换为连续的两个0xff

命令:

1. SET_SERIAL(设置串口参数) (0x01), 主机发

0xff	0x01	Mask	Mode	Flow	Ctrl	Baud
1	1	1	1	1	1	4

MASK :

Bit 0 MASK_BAUD

Bit 1 MASK_MODE

Bit 2 MASK_FLOW

Bit 3 MASK_BRK

Bit 4 MASK_CTRL

Bit 5 ~ bit 7 UNUSED

MODE : DATA BITS | STOP BITS | PARITY | BREAK

DATA BITS (bit 0, 1) = 0x00 = CS5

0x01 = CS6

0x02 = CS7

0x03 = CS8

STOP BITS (bit 2) = 0x00 = STOP1

0x04 = STOP2

or STOP1.5(only whendata bits is CS5)

PARITY (bit 3,4 5) = 0x00 = PAR_NONE

0x08 = PAR_ODD

0x18 = PAR_EVEN

0x28 = PAR_MARK

0x38 = PAR_SPACE

BREAK(bit 6) 0x00 = CLEAR BREAK
 0x40 = SET BREAK

bit7 UNUSED

FLOW :

- Bit 0 : OUT_CTS
- Bit 1 : OUT_DSR
- Bit 2 : OUT_X
- Bit 3 : OUT_XANY
- Bit 4 : IN_RTS
- Bit 5 : IN_DTR
- Bit 6 : IN_X
- Bit 7 : UNUSED

CTRL : DTR | RTS

- DTR (bit 0) 0 : OFF, 1 : ON
- RTS (bit 1) 0 : OFF, 1 : ON

BAUD : LOW WORD : high bits of real baud
 HIGH WORD : low bits of real baud

2. SET_REPORT(设置LIANDA 报告串口状态的方式) (0x02), 主机发

0xff	0x02	TYPE
1	1	1

TYPE :

- 0 : ONCE
- 1 : IF CHANGED
- 2 : TIMER (时间间隔是 1 second)

3. REPORT_SERIAL (LIANDA 设备报告状态) (0x03), 主机收

0xff	0x03	DATA
1	1	2

DATA :

1ST BYTE:

- BIT0 S_CTS
- BIT1 S_DSR
- BIT2 S_RI
- BIT3 S_DCD
- BIT4-7 UNUSED

2nd BYTE:

- BIT0 ERR_PARITY
- BIT1 ERR_FRAME
- BIT2 ERR_OVERRUN
- BIT3 ERR_BRK
- BIT4-7 UNUSED

4. CLOSE(主机端主机关闭连接是通知设备) (0x04) 主机发

0xff	0x04
------	------

5. TIME_BREAK(精确时间长度的BREAK 信号) (0x05) 主机发

0xff	0x05	TIMER
1	1	4

6. KEEP_ALIVE(主机端保活数据) 主机发

0xff	0xf1
------	------

五. 主机端的一些程序例子

1. 一些值的定义

```
/* LINE CONTROL */
/* DATA BITS DEFINE */
#define CS5 0x00
#define CS6 0x01
#define CS7 0x02
#define CS8 0x03

/* STOP BITS DEFINE */
#define STOP1 0x00
#define STOP15 0x04 // only when byte length is 5
#define STOP2 0x04

/* PARITY DEFINE */
#define PAR_NONE 0x00
#define PAR_ODD 0x08
#define PAR_EVEN 0x18
#define PAR_MARK 0x28
#define PAR_SPACE 0x38

/* MODEM CONTROL */
#define C_DTR 0x01
#define C_RTS 0x02

/* MODEM STATUS */
#define M_CTS 0x01
#define M_DSR 0x02
#define M_RI 0x04
#define M_CD 0x08

/* ERROR STATUS */
#define ERR_PARITY 0x01
#define ERR_FRAME 0x02
#define ERR_OVERRUN 0x04
#define ERR_BRK 0x08

/* FLOW CONTROL */
#define FLOW_NONE 0x00
#define OUT_CTS 0x01
#define OUT_DSR 0x02
#define OUT_X 0x04
#define OUT_XANY 0x08
#define IN_RTS 0x10
#define IN_DTR 0x20
#define IN_X 0x40

/* CTRL COMMAND */
#define NSER_CMD_SET_SERIAL 0x01
#define NSER_CMD_SET_REPORT 0x02
#define NSER_CMD_REPORT_SERIAL 0x03
#define NSER_CMD_CLOSE 0x04
```

```
#define NSER_CMD_TIME_BREAK 0x05

/* MASK DEFINE */
#define MASK_BAUD 0x01
#define MASK_MODE 0x02
#define MASK_FLOW 0x04
#define MASK_BRK 0x08
#define MASK_CTRL 0x10

/* REPORT SET */
#define REPORT_ONCE 0x00
#define REPORT_IF_CHANGED 0x01
#define REPORT_TIMER 0x02 // timer is 1 second
```

2. 建立连接

TCP SERVER模式下

```
#define NTD_IPADDR "192.168.0.233" // ip addr of LIANDA
#define NTD_PORT 1280 // sckcfg N -lt NTDPORT
```

```
int socket_fd;
struct sockaddr_in ntd_ip;
```

```
memset(&ntd_ip, 0, sizeof(ntd_ip));
ntd_ip.sin_family = AF_INET;
ntd_ip.sin_port = htons((unsigned short)NTD_PORT);
ntd_ip.sin_addr.s_addr = inet_addr(NTD_IPADDR);
```

```
socket_fd = socket(AF_INET, SOCK_STREAM, 0);
```

```
if (connect(socket_fd, (struct sockaddr *)&ntd_ip, sizeof(ntd_ip)) < 0)
{
    // error , do something and exit
}
else
{
    // succeed
}
```

TCP CLIENT模式下

```
#define LOCAL_IPADDR "192.168.0.1" // sckcfg N -rip
LOCAL_IPADDR
#define LOCAL_PORT 5000 // sckcfg N -rt LOCAL_PORT
```

```
int socket_fd, listen_fd;
```

```
struct sockaddr_in local_ip;
memset(&local_ip, 0, sizeof(local_ip));
local_ip.sin_family = AF_INET;
local_ip.sin_port = htons((unsigned short)LOCAL_PORT);
local_ip.sin_addr.s_addr = inet_addr(LOCAL_IPADDR);
```

```
listen_fd = socket(AF_INET, SOCK_STREAM, 0);
bind(listen_fd, (struct sockaddr *)&local_ip, sizeof(local_ip));
```

```
listen(listen_fd, 1);

socket_fd = accept(listen_fd, NULL, NULL);

if (socket_fd > 0)
{
    // succeed
}
else
{
    // error
}
```

UDP模式

```
#define LOCAL_IPADDR    "192.168.0.1"    // sckcfg N -rip LOCAL_IPADDR
#define LOCAL_PORT     5000             // sckcfg N -rt LOCAL_PORT
int                    udp_fd;
struct sockaddr_in local_ip;

memset(&local_ip, 0, sizeof(local_ip));
local_ip.sin_family = AF_INET;
local_ip.sin_port = htons((unsigned short)LOCAL_PORT);
local_ip.sin_addr.s_addr = inet_addr(LOCAL_IPADDR);

udp_fd = socket(AF_INET, SOCK_DGRAM, 0);
bind(udp_fd, (struct sockaddr *)&local_ip, sizeof(local_ip));

// do something other
```

3. 命令的发送(以TCP SOCKET 为例, UDP SOCKET 中把函数send改为sendto)

a. SET_SERIAL

```
void set_param(int fd)
{
    unsigned char mask;
    unsigned char mode;
    unsigned char flow;
    unsigned char ctrl;
    unsigned long baud;

    unsigned char buf[10];

    mode = CS8 | STOP1 | PAR_NONE;    // 8 bits data, 1 bit stop, no parity
    flow = FLOW_NONE;                // no flowctrl
    ctrl = C_DTR | C_RTS;            // DTR on, RTS on
    baud = 9600;                      // baud is 9600bps

    mask = MASK_MODE | MASK_FLOW | MASK_CTRL | MASK_BAUD;

    buf[0] = 0xff;
    buf[1] = NSER_CMD_SET_SERIAL;
```

```

buf[2] = mask;
buf[3] = mode;
buf[4] = flow;
buf[5] = ctrl;
*((unsigned short *)(buf + 6)) = (unsigned short)((baud >> 16) & 0xffff);
*((unsigned short *)(buf + 8)) = (unsigned short)(baud & 0xffff);

```

```

send(fd, (char *)buf, 10, 0);

```

```

}

```

b. SET_REPORT

```

void set_report(int fd)
{
    unsigned char type;
    unsigned char buf[3];

    type = REPORT_IF_CHANGED;

    buf[0] = 0xff;
    buf[1] = NSER_CMD_SET_REPORT;
    buf[2] = type;

    send(fd, (char *) buf, 3, 0);
}

```

c. TIME_BREAK

```

void send_time_break(int fd)
{
    unsigned char buf[6];
    unsigned long brktm;

    brktm = 1000; // 1 秒

    buf[0] = 0xff;
    buf[1] = NSER_CMD_TIME_BREAK;
    *((unsigned short *)(buf + 2)) = (unsigned short)((brktm >> 16) & 0xffff);
    *((unsigned short *)(buf + 4)) = (unsigned short)(brktm & 0xffff);
    send(fd, (char *)buf, 6, 0);
}

```

d. CLOSE

```

void send_close(int fd)
{
    unsigned char buf[2];
    buf[0] = 0xff;
    buf[1] = NSER_CMD_CLOSE;
    send(fd, (char *)buf, 2, 0);
}

```

e. KEEPALIVE

```

void send_keepalive(int fd)
{
    unsigned char buf[2];
    buf[0] = 0xff;
}

```

```

buf[1] = 0xf1;
send(fd, (char *)buf, 2, 0);
}

```

4. 数据的发送(以TCP SOCKET 为例, UDP SOCKET 中把函数send改为sendto)

注: fd是已建立好的SOCKET
 sendbuf 为待发送的数据缓冲
 sendlen 为待发送的数据长度

```

int i;
for (i = 0; i < sendlen; i++)
{
    if (sendbuf[i] == 0xff)
    {
        n = send(fd, "\0xff\0xff", 2, 0); // send two continuous 0xff
    }
    else
    {
        n = send(fd, sendbuf[i], 1, 0);
    }
}

```

5. 数据的接收(以TCP SOCKET 为例, UDP SOCKET 中把函数recv改为recvfrom)

注: fd是已建立好的SOCKET

```

char recvbuf[1024];
int recven, n;
int has_a_ff = 0; // if have receive a 0xff

recven = recv(fd, recvbuf, 1024, 0);
for (n = 0; n < recven; n++)
{
    if (recvbuf[n] == 0xff)
    {
        has_a_ff += 1;
        has_a_ff &= 0x01;
        if (has_a_ff == 0)
        {
            // 得到连续的两个0xff, 代表一个数据0xff
        }
    }
    if (has_a_ff)
    {
        switch(recvbuf[n])
        {
            case REPORT_SERIAL:
                // receive command REPORT_SERIAL
                break;
            default:
                break;
        }
    }
}
// DATA
}

```

