NAME

termios, tcgetattr, tcsetattr, tcsendbreak, tcdrain, tcflush, tcflow, cfmakeraw, cfgetospeed, cfgetispeed, cfsetispeed, cfsetospeed - 获取和设置终端属性，行控制，获取和设置波特率

SYNOPSIS 总览

#include <termios.h>
#include <unistd.h>

int tcgetattr(int *fd*, **struct termios** **termios_p*);

int tcsetattr(int *fd*, **int** *optional_actions*, **struct termios** **termios_p*);

int tcsendbreak(int *fd*, **int** *duration*);

int tcdrain(int *fd*);

int tcflush(int *fd*, **int** *queue_selector*);

int tcflow(int *fd*, **int** *action*);

int cfmakeraw(struct termios **termios_p*);

speed_t cfgetispeed(struct termios **termios_p*);

speed_t cfgetospeed(struct termios **termios_p*);

int cfsetispeed(struct termios **termios_p*, **speed_t** *speed*);

int cfsetospeed(struct termios **termios_p*, **speed_t** *speed*);

DESCRIPTION 描述

termios 函数族提供了一个常规的终端接口，用于控制非同步通信端口。

这里描述的大部分属性有一个 *termios_p* 类型的参数，它是指向一个 **termios** 结构的指针。这个结构包含了至少下列成员：

```
tcflag_t c_iflag;      /* 输入模式 */
tcflag_t c_oflag;      /* 输出模式 */
tcflag_t c_cflag;      /* 控制模式 */
tcflag_t c_lflag;      /* 本地模式 */
cc_t c_cc[NCCS];       /* 控制字符 */
```

c_iflag 标志常量：

IGNBRK
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 将视为与 NUL 字符同义，除非设置了 **PARMRK**，这种情况下它被视为序列 \377 \0 \0。
IGNPAR
忽略桢错误和奇偶校验错。
PARMRK
如果没有设置 **IGNPAR**，在有奇偶校验错或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**，将有奇偶校验错或桢错误的字符视为 \0。

INPCK
启用输入奇偶检测。
ISTRIP
去掉第八位。
INLCR
将输入中的 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 (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) 当输入队列满时响零。Linux 没有实现这一位，总是将它视为已设置。

POSIX.1 中定义的 *c_oflag* 标志常量：

OPOST
启用具体实现自行定义的输出处理。

其余 *c_oflag* 标志常量定义在 POSIX 1003.1-2001 中，除非另外说明。

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，而不是使用定时来延时。
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
回退延时掩码。取值为 **BS0** 或 **BS1**。(从来没有被实现过)
VTDLY
竖直跳格延时掩码。取值为 **VT0** 或 **VT1**。
FFDLY
进表延时掩码。取值为 **FF0** 或 **FF1**。

c_cflag 标志常量：

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。

(POSIX 规定波特率存储在 termios 结构中，并未精确指定它的位置，而是提供了函数 **cfgetispeed()** 和 **cfsetispeed()** 来存取它。一些系统使用 *c_cflag* 中 CBAUD 选择的位，其他系统使用单独的变量，例如 *sg_ispeed* 和 *sg_ospeed* 。)

CSIZE
字符长度掩码。取值为 **CS5**, **CS6**, **CS7**, 或 **CS8**。
CSTOPB
设置两个停止位，而不是一个。
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 **shl**)。(?)
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) 流控制。

c_lflag 标志常量：

ISIG
当接受到字符 INTR, QUIT, SUSP, 或 DSUSP 时，产生相应的信号。
ICANON
启用标准模式 (canonical mode)。允许使用特殊字符 EOF, EOL, EOL2, ERASE, KILL, LNEXT, REPRINT, STATUS, 和 WERASE，以及按行的缓冲。
XCASE
(不属于 POSIX; Linux 下不被支持) 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 A 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 (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 **ECHOPRT** 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 下不被支持) 输出被刷新。这个标志可以通过键入字符 DISCARD 来开关。

NOFLSH
禁止在产生 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 下不被支持) 在读入下一个字符时，输入队列中所有字符被重新输出。(**bash** 用它来处理 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 数组定义了特殊的控制字符。符号下标 (初始值) 和意义为：

VINTR
(003, ETX, Ctrl-C, or also 0177, DEL, rubout) 中断字符。发出 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。

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).

这些符号下标值是互不相同的，除了 VTIME，VMIN 的值可能分别与 VEOL，VEOF 相同。(在 non-canonical 模式下，特殊字符的含义更改为延时含义。MIN 表示应当被读入的最小字符数。TIME 是以十分之一秒为单位的计时器。如果同时设置了它们，read 将等待直到至少读入一个字符，一旦读入 MIN 个字符或者从上次读入字符开始经过了 TIME 时间就立即返回。如果只设置了 MIN，read 在读入 MIN 个字符之前不会返回。如果只设置了 TIME，read 将在至少读入一个字符，或者计时器超时的时候立即返回。如果都没有设置，read 将立即返回，只给出当前准备好的字符。) (?)

tcgetattr() 得到与 *fd* 指向的对象相关的参数，将它们保存于 *termios_p* 引用的 **termios** 结构中。函数可以从后台进程中调用；但是，终端属性可能被后来的前台进程所改变。

tcsetattr() 设置与终端相关的参数 (除非需要底层支持却无法满足)，使用 *termios_p* 引用的 **termios** 结构。*optional_actions* 指定了什么时候改变会起作用：

TCSANOW
改变立即发生

TCSADRAIN
改变在所有写入 *fd* 的输出都被传输后生效。这个函数应当用于修改影响输出的参数时使用。

TCSAFLUSH
改变在所有写入 *fd* 引用的对象的输出都被传输后生效，所有已接受但未读入的输入都在改变发生前丢弃。

tcsendbreak() 传送连续的 0 值比特流，持续一段时间，如果终端使用异步串行数据传输的话。如果 *duration* 是 0，它至少传输 0.25 秒，不会超过 0.5 秒。如果 *duration* 非零，它发送的时间长度由实现定义。

如果终端并非使用异步串行数据传输，**tcsendbreak()** 什么都不做。

tcdrain() 等待直到所有写入 *fd* 引用的对象的输出都被传输。

tcflush() 丢弃要写入 引用的对象，但是尚未传输的数据，或者收到但是尚未读取的数据，取决于 *queue_selector* 的值：

TCIFLUSH
刷新收到的数据但是不读

TCOFLUSH

刷新写入的数据但是不传送

TCIOFLUSH
同时刷新收到的数据但是不读，并且刷新写入的数据但是不传送

tcflow() 挂起 *fd* 引用的对象上的数据传输或接收，取决于 *action* 的值：

TCOOFF
挂起输出

TCOON
重新开始被挂起的输出

TCIOFF
发送一个 STOP 字符，停止终端设备向系统传送数据

TCION
发送一个 START 字符，使终端设备向系统传输数据

打开一个终端设备时的默认设置是输入和输出都没有挂起。

波特率函数被用来获取和设置 **termios** 结构中，输入和输出波特率的值。新值不会马上生效，直到成功调用了 **tcsetattr()** 函数。

设置速度为 **B0** 使得 modem "挂机"。与 **B38400** 相应的实际比特率可以用 **setserial**(8) 调整。

输入和输出波特率被保存于 **termios** 结构中。

cfmakeraw 设置终端属性如下：

```
termios_p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP
                            |INLCR|IGNCR|ICRNL|IXON);
            termios_p->c_oflag &= ~OPOST;
            termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN);
            termios_p->c_cflag &= ~(CSIZE|PARENB);
            termios_p->c_cflag |= CS8;
```

cfgetospeed() 返回 *termios_p* 指向的 **termios** 结构中存储的输出波特率

cfsetospeed() 设置 *termios_p* 指向的 **termios** 结构中存储的输出波特率为 *speed*。取值必须是以下常量之一：

B0
B50
B75
B110
B134
B150
B200
B300
B600
B1200
B1800
B2400
B4800
B9600
B19200
B38400
B57600
B115200
B230400

零值 **B0** 用来中断连接。如果指定了 B0，不应当再假定存在连接。通常，这样将断开连接

CBAUDEX 是一个掩码，指示高于 POSIX.1 定义的速度的那一些 (57600 及以上)。因此，**B57600** & **CBAUDEX** 为非零。

cfgetispeed() 返回 **termios** 结构中存储的输入波特率。

cfsetispeed() 设置 **termios** 结构中存储的输入波特率为 *speed*。如果输入波特率被设为*0，实际输入波特率将等于输出波特率。*

RETURN VALUE 返回值

cfgetispeed() 返回 **termios** 结构中存储的输入波特率。

cfgetospeed() 返回 **termios** 结构中存储的输出波特率。

其他函数返回：

0
成功
-1
失败，并且为 *errno* 置值来指示错误。

注意 **tcsetattr()** 返回成功，如果任何所要求的修改可以实现的话。因此，当进行多重修改时，应当在这个函数之后再次调用 **tcgetattr()** 来检测是否所有修改都成功实现。

NOTES 注意

Unix V7 以及很多后来的系统有一个波特率的列表，在十四个值 B0, ..., B9600 之后可以看到两个常数 EXTA, EXTB ("External A" and "External B")。很多系统将这个列表扩展为更高的波特率。

tcsendbreak 中非零的 *duration* 有不同的效果。SunOS 指定中断 *duration*N* 秒，其中 *N* 至少为 0.25，不高于 0.5 。Linux, AIX, DU, Tru64 发送 *duration* 微秒的 break 。FreeBSD, NetBSD, HP-UX 以及 MacOS 忽略 *duration* 的值。在 Solaris 和 Unixware 中， **tcsendbreak** 搭配非零的 *duration* 效果类似于 **tcdrain**。

所有的范例来源自 miniterm.c. The type ahead 暂存器被限制在 255 个字元, 就跟标准输入程序的最大字串长度相同 (<linux/limits.h> 或 <posix1_lim.h>).

参考程序码中的注解它会解释不同输入模式的使用. 我希望这些程序码都能被了解. 标准输入程序的程序范例的注解写得最好, 其它的范例都只在不同于其它范例的地方做注解.

叙述不是很完整, 但可以激励你对这范例做实验, 以延生出合于你所需应用程序的最佳解.

别忘记要把序列埠的权限设定正确 (也就是: chmod a+rw /dev/ttyS1)!

3.1 标准输入程序

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
/* 鲍率设定被定义在 <asm/termbits.h>, 这在 <termios.h> 被引入 */
#define BAUDRATE B38400
/* 定义正确的序列埠 */
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系统兼容 */
#define FALSE 0
#define TRUE 1
volatile int STOP=FALSE;
main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  char buf[255];
/*
  开启数据机装置以读取并写入而不以控制 tty 的模式
  因为我们不想程序在送出 CTRL-C 后就被杀掉.
*/
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }
 tcgetattr(fd,&oldtio); /* 储存目前的序列埠设定 */
 bzero(&newtio, sizeof(newtio)); /* 清除结构体以放入新的序列埠设定值 */
/*
  BAUDRATE: 设定 bps 的速度. 你也可以用 cfsetispeed 及 cfsetospeed 来设定.
  CRTSCTS : 输出资料的硬件流量控制 (只能在具完整线路的缆线下工作
            参考 Serial-HOWTO 第七节)
  CS8     : 8n1 (8 位元, 不做同位元检查,1 个终止位元)
  CLOCAL  : 本地连线, 不具数据机控制功能
  CREAD   : 致能接收字元
*/
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

/*
  IGNPAR  : 忽略经同位元检查后, 错误的位元组
  ICRNL   : 比 CR 对应成 NL (否则当输入信号有 CR 时不会终止输入)
            在不然把装置设定成 raw 模式(没有其它的输入处理)
*/
 newtio.c_iflag = IGNPAR | ICRNL;

/*
 Raw 模式输出.
*/
 newtio.c_oflag = 0;

/*
  ICANON  : 致能标准输入, 使所有回应机能停用, 并不送出信号以叫用程序
*/
 newtio.c_lflag = ICANON;

/*
  初始化所有的控制特性
```

```
   预设值可以在 /usr/include/termios.h 找到, 在注解中也有,
   但我们在这不需要看它们
*/
 newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
 newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
 newtio.c_cc[VERASE]   = 0;     /* del */
 newtio.c_cc[VKILL]    = 0;     /* @ */
 newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
 newtio.c_cc[VTIME]    = 0;     /* 不使用分割字元组的计时器 */
 newtio.c_cc[VMIN]     = 1;     /* 在读取到 1 个字元前先停止 */
 newtio.c_cc[VSWTC]    = 0;     /* '\0' */
 newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
 newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
 newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
 newtio.c_cc[VEOL]     = 0;     /* '\0' */
 newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
 newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
 newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
 newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
 newtio.c_cc[VEOL2]    = 0;     /* '\0' */
/*
   现在清除数据机线并启动序列埠的设定
*/
 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);
/*
   终端机设定完成, 现在处理输入信号
   在这个范例, 在一行的开始处输入 'z' 会退出此程序.
*/
 while (STOP==FALSE) {      /* 回圈会在我们发出终止的信号后跳出 */
 /* 即使输入超过 255 个字元, 读取的程序段还是会一直等到行终结符出现才停止.
    如果读到的字元组低于正确存在的字元组, 则所剩的字元会在下一次读取时取得.
    res 用来存放真正读到的字元组个数 */
    res = read(fd,buf,255);
    buf[res]=0;               /* 设定字串终止字元, 所以我们能用 printf */
    printf(":%s:%d\n", buf, res);
    if (buf[0]=='z') STOP=TRUE;
 }
 /* 回存旧的序列埠设定值 */
 tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.2 非标准输入程序

在非标准的输入程序模式下,输入的资料不会被组合成一行而输入后的处理功能 (清除,杀掉,删除,等等.) 都不能使用.这个模式有两个功能控制参数: c_cc[VTIME] 设定字元输入时间计时器,及 c_cc[VMIN] 设定满足读取功能的最低字元接收个数.

如果 MIN > 0 且 TIME = 0, MIN 设定为满足读取功能的最低字元接收个数.由于 TIME 是 零, 所以计时器将不被使用.

如果 MIN = 0 且 TIME > 0, TIME 将被当做逾时设定值.满足读取功能的情况为读取到单一
或者超过 TIME 所定义的时间 (t = TIME *0.1 s).如果超过 TIME 所定义的时间,则不会传回
字元.

如果 MIN > 0 且 TIME > 0, TIME 将被当做一个分割字元组的计时器. 满足读取功能的条件为接收到 MIN 个数的字元, 或两个字元的间隔时间超过 TIME 所定义的值. 计时器会在每读到一个字元后重新计时, 且只会在第一个字元收到后才会启动.

如果 MIN = 0 且 TIME = 0, 读取功能就马上被满足. 目前所存在的字元组个数, 或者 将回传的字元组个数. 根据 Antonino (参考 贡献) 所说, 你可以用 fcntl(fd, F_SETFL, FNDELAY); 在读取前得到相同的结果.

藉由修改 newtio.c_cc[VTIME] 及 newtio.c_cc[VMIN] 上述的模式就可以测试了.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系统兼容 */
#define FALSE 0
#define TRUE 1
volatile int STOP=FALSE;
main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  char buf[255];
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }
 tcgetattr(fd,&oldtio); /* 储存目前的序列埠设定 */
 bzero(&newtio, sizeof(newtio));
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_iflag = IGNPAR;
 newtio.c_oflag = 0;
 /* 设定输入模式 (非标准型, 不回应,...) */
 newtio.c_lflag = 0;

 newtio.c_cc[VTIME]    = 0;   /* 不使用分割字元组计时器 */
 newtio.c_cc[VMIN]     = 5;   /* 在读取到 5 个字元前先停止 */
 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);
 while (STOP==FALSE) {       /* 输入回圈 */
   res = read(fd,buf,255);   /* 在输入 5 个字元后即返回 */
   buf[res]=0;               /* 所以我们能用 printf... */
   printf(":%s:%d\n", buf, res);
   if (buf[0]=='z') STOP=TRUE;
 }
 tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.3 非同步式输入

```
#include <termios.h>
```

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>
#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系统兼容 */
#define FALSE 0
#define TRUE 1
volatile int STOP=FALSE;
void signal_handler_IO (int status);   /* 定义信号处理程序 */
int wait_flag=TRUE;                    /* 没收到信号的话就会是 TRUE */
main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  struct sigaction saio;           /* definition of signal action */
  char buf[255];
  /* 开启装置为 non-blocking (读取功能会马上结束返回) */
  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }
  /* 在使装置非同步化前, 安装信号处理程序 */
  saio.sa_handler = signal_handler_IO;
  saio.sa_mask = 0;
  saio.sa_flags = 0;
  saio.sa_restorer = NULL;
  sigaction(SIGIO,&saio,NULL);

  /* 允许行程去接收 SIGIO 信号*/
  fcntl(fd, F_SETOWN, getpid());
  /* 使文档ake the file descriptor 非同步 (使用手册上说只有 O_APPEND 及
  O_NONBLOCK, 而 F_SETFL 也可以用...) */
  fcntl(fd, F_SETFL, FASYNC);
  tcgetattr(fd,&oldtio); /* 储存目前的序列埠设定值 */
  /* 设定新的序列埠为标准输入程序 */
  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR | ICRNL;
  newtio.c_oflag = 0;
  newtio.c_lflag = ICANON;
  newtio.c_cc[VMIN]=1;
  newtio.c_cc[VTIME]=0;
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

  /* 等待输入信号的回圈. 很多有用的事我们将在这做 */
  while (STOP==FALSE) {
    printf(".\n");usleep(100000);
    /* 在收到 SIGIO 后, wait_flag = FALSE, 输入信号存在则可以被读取 */
    if (wait_flag==FALSE) {
      res = read(fd,buf,255);
      buf[res]=0;
      printf(":%s:%d\n", buf, res);
      if (res==1) STOP=TRUE; /* 如果只输入 CR 则停止回圈 */
      wait_flag = TRUE;      /* 等待新的输入信号 */
```

```
    }
  }
  /* 回存旧的序列埠设定值 */
  tcsetattr(fd,TCSANOW,&oldtio);
}
/***************************************************************************
* 信号处理程序. 设定 wait_flag 为 FALSE, 以使上述的回圈能接收字元          *
***************************************************************************/
void signal_handler_IO (int status)
{
  printf("received SIGIO signal.\n");
  wait_flag = FALSE;
}
```

3.4 等待来自多个信号来源的输入

这一段很短.它只能被拿来当成写程序时的提示,故范例程序也很简短.但这个范例不只能用在序列埠上,还可以用在被当成文档来使用的装置上.

select 呼叫及伴随它所引发的巨集共用 fd_set.fd_set 则是一个位元阵列,而其中每一个位元代表一个有效的文档叙述结构.select 呼叫接受一个有效的文档叙述结构并传回 fd_set 位元阵列,而该位元阵列中若有某一个位元为 1,就表示相对映的文档叙述结构的文档发生了输入,输出或有例外事件.而这些巨集提供了所有处理 fd_set 的功能.亦可参考手册 select(2).

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
main()
{
   int    fd1, fd2;  /* 输入源 1 及 2 */
   fd_set readfs;    /* 文档叙述结构设定 */
   int    maxfd;     /* 最大可用的文档叙述结构 */
   int    loop=1;    /* 回圈在 TRUE 时成立 */
   /* open_input_source 开启一个装置, 正确的设定好序列埠,
      并回传回此文档叙述结构体 */
   fd1 = open_input_source("/dev/ttyS1");   /* COM2 */
   if (fd1<0) exit(0);
   fd2 = open_input_source("/dev/ttyS2");   /* COM3 */
   if (fd2<0) exit(0);
   maxfd = MAX (fd1, fd2)+1;  /* 测试最大位元输入 (fd) */
   /* 输入回圈 */
   while (loop) {
     FD_SET(fd1, &readfs);  /* 测试输入源 1 */
     FD_SET(fd2, &readfs);  /* 测试输入源 2 */
     /* block until input becomes available */
     select(maxfd, &readfs, NULL, NULL, NULL);
     if (FD_ISSET(fd1))         /* 如果输入源 1 有信号 */
       handle_input_from_source1();
     if (FD_ISSET(fd2))         /* 如果输入源 2 有信号 */
       handle_input_from_source2();
   }
}
```

这个范例程序在等待输入信号出现前,不能确定它会停顿下来.如果你需要在输入时加入逾时

能, 只需把 select 呼叫换成:

```
int res;
struct timeval Timeout;
/* 设定输入回圈的逾时值 */
Timeout.tv_usec = 0;  /* 毫秒 */
Timeout.tv_sec  = 1;  /* 秒 */
res = select(maxfd, &readfs, NULL, NULL, &Timeout);
if (res==0)
/* 文档叙述结构数在 input = 0 时, 会发生输入逾时. */
```

这个程序会在 1 秒钟后逾时. 如果超过时间, select 会传回 0, 但是应该留意 Timeout 的时间递减是由 select 所等待输入信号的时间为基准. 如果逾时的值是 0, select 会马上结束返回.

Linux 环境下使用RS-232接口

RS是英文 "推荐标准"的缩写
232为标识号
RS-485

串口通信表示计算机一次传送一个位的数据，
当使用串行通信时，每个字的数据是一个位一个位的传输或接收的，
每个位不是高电平，就是低电平.

串行通信的速率通常是使用"位/每秒"的方式来表示的，即波特率。

全双工--计算机可以同时收发数据,
它有两个独立的数据通道，一个输入，一个输出，

半双工意味着计算机不能同时收发信息，
只能有一人通道进行通信.

流控:

通常，当数据在两个串行接口之间进行传输时需要对其进行控制.
这通常依赖于串行通信连接的各种规定，

对异步数据传输的控制有两种方法.

一种叫：“软件”流控 。
一种叫： “硬件"流控 。

串口设备：

打开一个串行口

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h> // 文件控制定义
#include <errno.h>
#include <termios.h> //POSIX终端控制定义

/*
```

```
* open_port() --打开串行口
*
* 成功的话，返回文件描述符，错误则返回 -1.
*/

int open_port(void)
{
   int fd;
   fd=open("/dev/ttyS0",O_RDWR|O_NOCTTY|O_NDELAY);
   if (fd == -1)
   {
   /*无法打开串口*/
   perror("open_port : Unable to open /dev/ttyS0");
   }
   else
      fcntl(fd,F_SETFL,0);
   return (fd);
}

//O_NOCTTY 标志 ， 该程序不想成为此端口的“控制终端"。
              如果没有强调这一点，
//O_NDELAY标志 , 标志告诉Linux ,该程序并不关注DCD信叼线所处的状态,
即不管另外一端的设备是在运行还是被挂起。如果没有指定该标志，那么程序就会被设置睡
眠状态，
```

（2）向端口写数据

向端口写数据是很容易的，只要使用write()系统调用就可以了。

例如:

```
    n=write(fd,"ATZ\r",4);
if (n<0)
   fputs("write() of 4 bytes failed!\n",stderr);
```

write函数返回发送数据的个数，如果出现错误，则返回 -1。

(3) 读端口数据

从端口读数据则需要些技巧。如果在原始数据的模式下对端口进行操作,
read()系统调用将返回串行口输入缓冲区中所有的字符数据，不管有多少，

如果没有数据，那么该调用将被阻塞.处于等待状态，直到有字符输入，
或者到了规定的时限和出现错误为止,
通过以下方法，能使read函数立即返回。

```
fcntl(fd,F_SETFL,FNDELAY);
```

FNDELAY 函数使read函数在端口没月字符存在的情况下，立刻返回0,
如果要恢复正常(阻塞)状态,可以调用fcntl()函数，不要FNDELAY参数,
如下所示：

```
   fcntl(Fd,F_SETFL,0);
```

在使用O_NDELAY参数打开串行口后，同样与使用了该函数调用。

```
fcntl(fd,F_SETFL,0);
```

关键字：串口资料