NAME

vtp − virtual terminal protocol

DESCRIPTION

This section describes how to use the virtual terminal protocol feature that can be optionally enabled in the operating system.

The virtual terminal protocol provides a means whereby user programs can be written to interact with CRT terminals in a language which is independent of the actual type of terminal. The operating system provides translation of standard sequences of characters into the sequences necessary to cause the desired behavior on each type of terminal. It also translates what the terminal sends to the system so that the user program sees the standard sequence for all terminals regardless of the terminal actually in use.

An *ioctl*() is necessary to enable a specific terminal handler and that terminal handler must have been compiled into the operating system. The *ioctl* call is described in <**sys/termio.h**>:

```
/*
 * structure of ioctl arg for LDGETT and LDSETT
 */
struct      termcb      {
            char        st_flgs;    /* term flags */
            char        st_termt;   /* term type */
            char        st_crow;    /* gtty only - current row */
            char        st_ccol;    /* gtty only - current col */
            char        st_vrow;    /* variable row */
            char        st_lrow;    /* gtty only - last row */
};
```

Terminals for which drivers are currently available are the DEC vt61 and vt100, the TEC scope, the Teletype D40, the Hewlett Packard hp26xx terminals, and the Concept 100.

The terminal flags are automatically set by the *ioctl*() on a **LDSETT** command to appropriate values for a specific terminal unless the user overrides these defaults by setting the **TM_SET** bit. If the user does this, then the terminal flags are set according to the other flags found in "st_flgs".

TM_SNL The special newline flag means that the newline character will be treated specially. Currently this is used by the Dataspeed 40 terminals. When this flag is set, newline characters are converted to "load cursor address sequences" so that the printed newline character doesn't appear on the screen.

TM_ANL Causes an automatic newline whenever the cursor attempts to pass the 80th column.

TM_LCF The "last column function" flag causes scrolling to be emulated on dumb terminals, such as the TEC, which do not have scrolling hardware. Scrolling is emulated by moving the cursor to the first row of the terminal, deleting the line, and then moving the cursor to the bottom of the screen again.

TM_CECHO

Causes the cursor motion keys to function without user software intervention by causing the codes generated by the cursor control keys to be immediately echoed back to the terminal when they are received. This flag is usually used in conjunction with the **TM_CINVIS** flag.

TM_CINVIS

Inhibits the translation and tranmission of the cursor motion sequences to the user program. If this flag and the **TM_CECHO** flag are on, that the cursor

motion keys work without intervention by the user process.

TM_SET Causes tne values of the preceeding flags to be set or cleared if the **LDSETT** *ioctl*() command is being done.

"st_vrow" specifies the row at which scrolling will take place. This means that everything on the screen above that row will be unaffected as material scrolls upwards from the bottom. This allows split screen operation. "st_crow" and "st_ccol" contain the system's idea of the current row and column when a **LDGETT** command is done. "st_lrow" contains the system's idea of which row is the last row visible on the CRT screen. To assure that the system and the terminal both agree on the cursor position, a **VHOME** escape sequence should be transmitted to the terminal after the terminal handler is enabled. Columns and rows are numbered from (0,0).

Once the terminal type is set, user programs use the escape sequences described in **/usr/include/sys/crtctl.h** to control the behavior of the terminal.

```
/*              @(#)crtctl.h    3.2             */

/*
                Define the cursor control codes
*/
#define ESC     033             /* Escape for command */

/* Commands */
#define CUP     0101            /* Cursor up */
#define CDN     0102            /* Cursor down */
#define CRI     0103            /* Cursor right */
#define CLE     0104            /* Cursor left */
#define HOME    0105            /* Cursor home */
#define VHOME   0106            /* cursor home to variable portion */
#define LCA     0107            /* Load cursor, followed by (x,y) in (col,row) */

#define STB     0110            /* Start blink */
#define SPB     0111            /* Stop blink */
#define CS      0112            /* Clear Screen */
#define EEOL    0113            /* Erase to end of line */
#define EEOP    0114            /* Erase to end of page */
#define DC      0115            /* Delete character */
#define DL      0116            /* Delete line */
#define IC      0117            /* Insert character */
#define IL      0120            /* Insert line */
#define KBL     0121            /* keyboard lock */
#define KBU     0122            /* keyboard unlock */
#define ATAB    0123            /* Set Column of tabs on all lines */
#define STAB    0124            /* Set single tab on current line only */
#define CTAB    0125            /* Clear all tabs */
#define CSTAB   0144            /* Clear tab at current column, all lines */
#define USCRL   0126            /* Scroll up one line */
#define DSCRL   0127            /* Scroll down one line */
#define ASEG    0130            /* Advance segment */
#define BPRT    0131            /* Begin protect */
#define EPRT    0132            /* End  protect */

#define CRTN    0133            /* Return cursor to beginning of line */
#define NL      0134            /* Terminal newline function */
#define CM      0135            /* Clear Memory (Terminal Reset) */
#define SVSCN   0136            /* Define variable portion of screen (OS only) */
#define UVSCN   0137            /* Scroll Up variable portion of screen */
#define DVSCN   0140            /* Scroll Down variable portion of screen */
```

```
#define SVID    0141        /* Set Video Attributes */
#define CVID    0142        /* Clear Video Attributes */
#define DVID    0143        /* Define Video Attributes */
/* Video Attribute Definitions */
#define VID_NORM            000         /* normal */
#define VID_UL              001         /* underline */
#define VID_BLNK            002         /* blink */
#define VID_REV             004         /* reverse video */
#define VID_DIM             010         /* dim intensity */
#define VID_BOLD            020         /* bright intensity */
#define VID_OFF             040         /* blank out field */

#define BRK     000         /* transmit break */
#define HIQ     001         /* Put remainder of this write on the high
                               priority queue, saving current cursor and restoring
                               when done. */
```

When sending escape sequences to the terminal, it is necessary that the writes be atomic. In other words, if it is desired to move the cursor to column 10 and row 5, it is necessary to send the four characters, ESC LCA 10 5, to the terminal in a single write system call. If standard I/O is being used, the stream to the terminal must be buffered, and then flushed after the escape sequence is written so that the write is atomic.

Note that some of these functions will not work with every terminal. Whether a function works or not is dependent on how smart the terminal handler code in the operating system is and what capabilities the terminal itself has. Basically you can expect that all terminal handlers can manage the cursor motion commands, CUP, CDN, CRI, CLE, HOME, VHOME, and LCA. Most terminal handlers can also do the scrolling of the variable portion of the screen, UVSCN and DVSCN, though sometimes the terminal handler emulates scrolling by deleting a line at the top of the region and then writing a new one at the bottom.

If a terminal has advanced video features such as blinking, underlining, and reverse video, it is possible to turn these features on and off with the SVID, CVID, and DVID commands. The "set video attributes", SVID, logically ors in the features specified by the next character. "Clear video attributes", CVID, and complements out the features specified by the next character. "Define video attributes", DVID, replaces the current video attributes with those specified by the next character.

Of particular utility is the "hi-queue write", HIQ. When combined with the variable scroll feature, it is possible to prevent some section at the top of the screen from being changed by normal writes and then have a special program perform hi-queue writes, which contain a "load cursor address" function (LCA) to modify the contents at the top of the screen. This allows the possibility of having a background process keep a display at the top of the screen updated while the user continues working in the lower portion of the screen. Hi-queue writes are limited to 512 bytes.

DEFICIENCIES

One annoying fact is that the HIQ string is terminated by the end of the write system call.