NAME

ioctl_console − ioctls for console terminal and virtual consoles

DESCRIPTION

The following Linux-specific **ioctl**(2) requests are supported for console terminals and virtual consoles. Each requires a third argument, assumed here to be *argp*.

KDGETLED

Get state of LEDs. *ar gp* points to a *char*. The lower three bits of**ar gp* are set to the state of the LEDs, as follows:

LED_CAP	0x04	caps lock led
LED_NUM	0x02	num lock led
LED_SCR	0x01	scroll lock led

KDSETLED

Set the LEDs. The LEDs are set to correspond to the lower three bits of the unsigned long integer in *argp*. However, if a higher order bit is set, the LEDs revert to normal: displaying the state of the keyboard functions of caps lock, num lock, and scroll lock.

Before Linux 1.1.54, the LEDs just reflected the state of the corresponding keyboard flags, and KDGETLED/KDSETLED would also change the keyboard flags. Since Linux 1.1.54 the LEDs can be made to display arbitrary information, but by default they display the keyboard flags. The following two ioctls are used to access the keyboard flags.

KDGKBLED

Get keyboard flags CapsLock, NumLock, ScrollLock (not lights). *ar gp* points to a char which is set to the flag state. The low order three bits (mask 0x7) get the current flag state, and the low order bits of the next nibble (mask 0x70) get the default flag state. (Since Linux 1.1.54.)

KDSKBLED

Set keyboard flags CapsLock, NumLock, ScrollLock (not lights). *argp* is an unsigned long integer that has the desired flag state. The low order three bits (mask 0x7) have the flag state, and the low order bits of the next nibble (mask 0x70) have the default flag state. (Since Linux 1.1.54.)

KDGKBTYPE

Get keyboard type. This returns the value KB_101, defined as 0x02.

KDADDIO

Add I/O port as valid. Equivalent to *ioperm(arg,1,1)*.

KDDELIO

Delete I/O port as valid. Equivalent to *ioperm(arg,1,0)*.

KDENABIO

Enable I/O to video board. Equivalent to *ioperm(0x3b4, 0x3df−0x3b4+1, 1)*.

KDDISABIO

Disable I/O to video board. Equivalent to *ioperm(0x3b4, 0x3df−0x3b4+1, 0)*.

KDSETMODE

Set text/graphics mode. *argp* is an unsigned integer containing one of:

KD_TEXT	0x00
KD_GRAPHICS	0x01

KDGETMODE

Get text/graphics mode. *ar gp* points to an *int* which is set to one of the values shown above for **KDSETMODE**.

KDMKTONE

Generate tone of specified length. The lower 16 bits of the unsigned long integer in *argp* specify the period in clock cycles, and the upper 16 bits give the duration in msec. If the duration is zero, the sound is turned off. Control returns immediately. For example, *argp* = (125<<16) + 0x637

would specify the beep normally associated with a ctrl-G. (Thus since Linux 0.99pl1; broken in Linux 2.1.49-50.)

KIOCSOUND
Start or stop sound generation. The lower 16 bits of *argp* specify the period in clock cycles (that is, *argp* = 1193180/frequency). *argp* = 0 turns sound off. In either case, control returns immediately.

GIO_CMAP
Get the current default color map from kernel. *argp* points to a 48-byte array. (Since Linux 1.3.3.)

PIO_CMAP
Change the default text-mode color map. *ar gp* points to a 48-byte array which contains, in order, the Red, Green, and Blue values for the 16 available screen colors: 0 is off, and 255 is full intensity. The default colors are, in order: black, dark red, dark green, brown, dark blue, dark purple, dark cyan, light grey, dark grey, bright red, bright green, yellow, bright blue, bright purple, bright cyan, and white. (Since Linux 1.3.3.)

GIO_FONT
Gets 256-character screen font in expanded form. *ar gp* points to an 8192-byte array. Fails with error code **EINVAL** if the currently loaded font is a 512-character font, or if the console is not in text mode.

GIO_FONTX
Gets screen font and associated information. *ar gp* points to a *struct consolefontdesc* (see **PIO_FONTX**). On call, the*c harcount* field should be set to the maximum number of characters that would fit in the buffer pointed to by *chardata*. On return, the*c harcount* and *charheight* are filled with the respective data for the currently loaded font, and the *chardata* array contains the font data if the initial value of *charcount* indicated enough space was available; otherwise the buffer is untouched and *errno* is set to **ENOMEM**. (Since Linux 1.3.1.)

PIO_FONT
Sets 256-character screen font. Load font into the EGA/VGA character generator. *argp* points to an 8192-byte map, with 32 bytes per character. Only the first*N* of them are used for an 8x*N* font ($0 < N <= 32$). This call also invalidates the Unicode mapping.

PIO_FONTX
Sets screen font and associated rendering information. *argp* points to a

```
struct consolefontdesc {
    unsigned short charcount;  /* characters in font
                                  (256 or 512) */
    unsigned short charheight; /* scan lines per
                                  character (1–32) */
    char          *chardata;   /* font data in
                                  expanded form */
};
```

If necessary, the screen will be appropriately resized, and **SIGWINCH** sent to the appropriate processes. This call also invalidates the Unicode mapping. (Since Linux 1.3.1.)

PIO_FONTRESET
Resets the screen font, size, and Unicode mapping to the bootup defaults. *argp* is unused, but should be set to NULL to ensure compatibility with future versions of Linux. (Since Linux 1.3.28.)

GIO_SCRNMAP
Get screen mapping from kernel. *argp* points to an area of size E_TABSZ, which is loaded with the font positions used to display each character. This call is likely to return useless information if the currently loaded font is more than 256 characters.

GIO_UNISCRNMAP
Get full Unicode screen mapping from kernel. *argp* points to an area of size *E_TABSZ*sizeof(unsigned short)*, which is loaded with the Unicodes each character represent. A special set of Unicodes, starting at U+F000, are used to represent "direct to font" mappings. (Since Linux 1.3.1.)

PIO_SCRNMAP
Loads the "user definable" (fourth) table in the kernel which maps bytes into console screen symbols. *argp* points to an area of size E_TABSZ.

PIO_UNISCRNMAP
Loads the "user definable" (fourth) table in the kernel which maps bytes into Unicodes, which are then translated into screen symbols according to the currently loaded Unicode-to-font map. Special Unicodes starting at U+F000 can be used to map directly to the font symbols. (Since Linux 1.3.1.)

GIO_UNIMAP
Get Unicode-to-font mapping from kernel. *argp* points to a

```
struct unimapdesc {
    unsigned short  entry_ct;
    struct unipair *entries;
};
```

where *entries* points to an array of

```
struct unipair {
    unsigned short unicode;
    unsigned short fontpos;
};
```

(Since Linux 1.1.92.)

PIO_UNIMAP
Put unicode-to-font mapping in kernel. *argp* points to a *struct unimapdesc*. (Since Linux 1.1.92)

PIO_UNIMAPCLR
Clear table, possibly advise hash algorithm. *argp* points to a

```
struct unimapinit {
    unsigned short advised_hashsize;  /* 0 if no opinion */
    unsigned short advised_hashstep;  /* 0 if no opinion */
    unsigned short advised_hashlevel; /* 0 if no opinion */
};
```

(Since Linux 1.1.92.)

KDGKBMODE
Gets current keyboard mode. *argp* points to a *long* which is set to one of these:

```
K_RAW       0x00  /* Raw (scancode) mode */
K_XLATE     0x01  /* Translate keycodes using keymap */
K_MEDIUMRAW 0x02  /* Medium raw (scancode) mode */
K_UNICODE   0x03  /* Unicode mode */
K_OFF       0x04  /* Disabled mode; since Linux 2.6.39 */
```

KDSKBMODE
Sets current keyboard mode. *argp* is a *long* equal to one of the values shown for **KDGKBMODE**.

KDGKBMETA
Gets meta key handling mode. *argp* points to a *long* which is set to one of these:

```
K_METABIT   0x03   set high order bit
K_ESCPREFIX 0x04   escape prefix
```

KDSKBMETA
Sets meta key handling mode. *ar gp* is a *long* equal to one of the values shown above for **KDGKBMETA**.

KDGKBENT
Gets one entry in key translation table (keycode to action code). *argp* points to a

```
struct kbentry {
    unsigned char  kb_table;
    unsigned char  kb_index;
    unsigned short kb_value;
};
```

with the first two members filled in: *kb_table* selects the key table (0 <= *kb_table* < MAX_NR_KEYMAPS), and *kb_index* is the keycode (0 <= *kb_index* < NR_KEYS). *kb_value* is set to the corresponding action code, or K_HOLE if there is no such key, or K_NOSUCHMAP if *kb_table* is invalid.

KDSKBENT
Sets one entry in translation table. *argp* points to a *struct kbentry*.

KDGKBSENT
Gets one function key string. *argp* points to a

```
struct kbsentry {
    unsigned char kb_func;
    unsigned char kb_string[512];
};
```

kb_string is set to the (null-terminated) string corresponding to the *kb_func*th function key action code.

KDSKBSENT
Sets one function key string entry. *argp* points to a *struct kbsentry*.

KDGKBDIACR
Read kernel accent table. *argp* points to a

```
struct kbdiacrs {
    unsigned int   kb_cnt;
    struct kbdiacr kbdiacr[256];
};
```

where *kb_cnt* is the number of entries in the array, each of which is a

```
struct kbdiacr {
    unsigned char diacr;
    unsigned char base;
    unsigned char result;
};
```

KDGETKEYCODE
Read kernel keycode table entry (scan code to keycode). *argp* points to a

```
struct kbkeycode {
    unsigned int scancode;
    unsigned int keycode;
};
```

keycode is set to correspond to the given *scancode*. (89 <=*scancode* <= 255 only . For 1 <= *scancode* <= 88, *keycode*==*scancode*.) (Since Linux 1.1.63.)

KDSETKEYCODE
Write kernel keycode table entry. *argp* points to a *struct kbkeycode*. (Since Linux 1.1.63.)

KDSIGACCEPT
The calling process indicates its willingness to accept the signal *argp* when it is generated by pressing an appropriate key combination. (1 <= *argp* <= NSIG). (See *spawn_console*() in *linux/drivers/char/keyboard.c*.)

VT_OPENQRY
Returns the first available (non-opened) console. *ar gp* points to an *int* which is set to the number of the vt (1 <=**ar gp* <= MAX_NR_CONSOLES).

VT_GETMODE
Get mode of active vt. *argp* points to a

```
struct vt_mode {
    char  mode;    /* vt mode */
    char  waitv;   /* if set, hang on writes if not active */
    short relsig;  /* signal to raise on release req */
    short acqsig;  /* signal to raise on acquisition */
    short frsig;   /* unused (set to 0) */
};
```

which is set to the mode of the active vt. *mode* is set to one of these values:

VT_AUTO	auto vt switching
VT_PROCESS	process controls switching
VT_ACKACQ	acknowledge switch

VT_SETMODE
Set mode of active vt. *argp* points to a *struct vt_mode*.

VT_GETSTATE
Get global vt state info. *argp* points to a

```
struct vt_stat {
    unsigned short v_active;  /* active vt */
    unsigned short v_signal;  /* signal to send */
    unsigned short v_state;   /* vt bit mask */
};
```

For each vt in use, the corresponding bit in the *v_state* member is set. (Linux 1.0 through Linux 1.1.92.)

VT_RELDISP
Release a display.

VT_ACTIVATE
Switch to vt *argp* (1 <= *argp* <= MAX_NR_CONSOLES).

VT_WAITACTIVE
Wait until vt *argp* has been activated.

VT_DISALLOCATE
Deallocate the memory associated with vt *argp*. (Since Linux 1.1.54.)

VT_RESIZE
Set the kernel's idea of screensize. *argp* points to a

```
struct vt_sizes {
    unsigned short v_rows;       /* # rows */
    unsigned short v_cols;       /* # columns */
    unsigned short v_scrollsize; /* no longer used */
};
```

Note that this does not change the videomode. See **resizecons**(8). (Since Linux 1.1.54.)

VT_RESIZEX
Set the kernel's idea of various screen parameters. *argp* points to a

```
struct vt_consize {
    unsigned short v_rows;  /* number of rows */
    unsigned short v_cols;  /* number of columns */
    unsigned short v_vlin;  /* number of pixel rows
                               on screen */
    unsigned short v_clin;  /* number of pixel rows
                               per character */
    unsigned short v_vcol;  /* number of pixel columns
                               on screen */
    unsigned short v_ccol;  /* number of pixel columns
                               per character */
};
```

Any parameter may be set to zero, indicating "no change", but if multiple parameters are set, they must be self-consistent. Note that this does not change the videomode. See **resizecons**(8). (Since Linux 1.3.3.)

The action of the following ioctls depends on the first byte in the struct pointed to by *argp*, referred to here as the *subcode*. These are legal only for the superuser or the owner of the current terminal. Symbolic *subcode*s are available in<*linux/tiocl.h*> since Linux 2.5.71.

TIOCLINUX, **subcode**=**0**
Dump the screen. Disappeared in Linux 1.1.92. (With Linux 1.1.92 or later, read from */dev/vcsN* or */dev/vcsaN* instead.)

TIOCLINUX, **subcode**=**1**
Get task information. Disappeared in Linux 1.1.92.

TIOCLINUX, **subcode**=**TIOCL_SETSEL**
Set selection. *argp* points to a

```
struct {
    char  subcode;
    short xs, ys, xe, ye;
    short sel_mode;
};
```

xs and *ys* are the starting column and row. *xe* and *ye* are the ending column and row. (Upper left corner is row=column=1.) *sel_mode* is 0 for character-by-character selection, 1 for word-by-word selection, or 2 for line-by-line selection. The indicated screen characters are highlighted and saved in a kernel buffer.

TIOCLINUX, **subcode**=**TIOCL_PASTESEL**
Paste selection. The characters in the selection buffer are written to *fd*.

TIOCLINUX, **subcode**=**TIOCL_UNBLANKSCREEN**
Unblank the screen.

TIOCLINUX, **subcode**=**TIOCL_SELLOADLUT**
Sets contents of a 256-bit look up table defining characters in a "word", for word-by-word selection. (Since Linux 1.1.32.)

TIOCLINUX, **subcode**=**TIOCL_GETSHIFTSTATE**
argp points to a char which is set to the value of the kernel variable *shift_state*. (Since Linux 1.1.32.)

TIOCLINUX, **subcode**=**TIOCL_GETMOUSEREPORTING**
argp points to a char which is set to the value of the kernel variable *report_mouse*. (Since Linux 1.1.33.)

TIOCLINUX, **subcode**=**8**
Dump screen width and height, cursor position, and all the character-attribute pairs. (Linux 1.1.67 through Linux 1.1.91 only. With Linux 1.1.92 or later, read from */dev/vcsa** instead.)

TIOCLINUX, **subcode**=**9**
Restore screen width and height, cursor position, and all the character-attribute pairs. (Linux 1.1.67 through Linux 1.1.91 only. With Linux 1.1.92 or later, write to */dev/vcsa** instead.)

TIOCLINUX, **subcode**=**TIOCL_SETVESABLANK**
Handles the Power Saving feature of the new generation of monitors. VESA screen blanking mode is set to *argp[1]*, which governs what screen blanking does:

0 Screen blanking is disabled.

1 The current video adapter register settings are saved, then the controller is programmed to turn off the vertical synchronization pulses. This puts the monitor into "standby" mode. If your monitor has an Off_Mode timer, then it will eventually power down by itself.

2 The current settings are saved, then both the vertical and horizontal synchronization pulses are turned off. This puts the monitor into "off" mode. If your monitor has no Off_Mode timer, or if you want your monitor to power down immediately when the blank_timer times out, then you choose this option. (*Caution:* Powering down frequently will damage the monitor.) (Since Linux 1.1.76.)

TIOCLINUX, **subcode**=**TIOCL_SETKMSGREDIRECT**
Change target of kernel messages ("console"): by default, and if this is set to **0**, messages are written to the currently active VT. The VT to write to is a single byte following **subcode**. (Since Linux 2.5.36.)

TIOCLINUX, **subcode**=**TIOCL_GETFGCONSOLE**
Returns the number of VT currently in foreground. (Since Linux 2.5.36.)

TIOCLINUX, **subcode**=**TIOCL_SCROLLCONSOLE**
Scroll the foreground VT by the specified amount of *lines* down, or half the screen if **0**. *lines* is *(((int32_t *)&subcode) + 1). (Since Linux 2.5.67.)

TIOCLINUX, **subcode**=**TIOCL_BLANKSCREEN**
Blank the foreground VT, ignoring "pokes" (typing): can only be unblanked explicitly (by switching VTs, to text mode, etc.). (Since Linux 2.5.71.)

TIOCLINUX, **subcode**=**TIOCL_BLANKEDSCREEN**
Returns the number of VT currently blanked, **0** if none. (Since Linux 2.5.71.)

TIOCLINUX, **subcode**=**16**
Never used.

TIOCLINUX, **subcode**=**TIOCL_GETKMSGREDIRECT**
Returns target of kernel messages. (Since Linux 2.6.17.)

RETURN VALUE

On success, 0 is returned (except where indicated). On failure, −1 is returned, and *errno* is set to indicate the error.

ERRORS

EBADF
The file descriptor is invalid.

EINVAL
The file descriptor or *argp* is invalid.

ENOTTY
The file descriptor is not associated with a character special device, or the specified request does not apply to it.

EPERM
Insufficient permission.

NOTES

Warning: Do not regard this man page as documentation of the Linux console ioctls. This is provided for the curious only, as an alternative to reading the source. Ioctl's are undocumented Linux internals, liable to be changed without warning. (And indeed, this page more or less describes the situation as of kernel version 1.1.94; there are many minor and not-so-minor differences with earlier versions.)

Very often, ioctls are introduced for communication between the kernel and one particular well-known program (fdisk, hdparm, setserial, tunelp, loadkeys, selection, setfont, etc.), and their behavior will be changed when required by this particular program.

Programs using these ioctls will not be portable to other versions of UNIX, will not work on older versions of Linux, and will not work on future versions of Linux.

Use POSIX functions.

SEE ALSO

dumpkeys(1), **kbd_mode**(1), **loadkeys**(1), **mknod**(1), **setleds**(1), **setmetamode**(1), **execve**(2), **fcntl**(2), **ioctl_tty**(2), **ioperm**(2), **termios**(3), **console_codes**(4), **mt**(4), **sd**(4), **tty**(4), **ttyS**(4), **vcs**(4), **vcsa**(4), **charsets**(7), **mapscrn**(8), **resizecons**(8), **setfont**(8)

/usr/include/linux/kd.h, */usr/include/linux/vt.h*