NAME

ioctl − control device

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int *fd***, unsigned long** *r equest***, ...);**

DESCRIPTION

The **ioctl**() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with **ioctl**() requests. The argument *fd* must be an open file descriptor.

The second argument is a device-dependent request code. The third argument is an untyped pointer to memory. It’s traditionally **char ****argp* (from the days before **void *** was valid C), and will be so named for this discussion.

An **ioctl**() *request* has encoded in it whether the argument is an *in* parameter or *out* parameter, and the size of the argument *argp* in bytes. Macros and defines used in specifying an **ioctl**() *request* are located in the file <*sys/ioctl.h*>. See NOTES.

RETURN VALUE

Usually, on success zero is returned. A few **ioctl**() requests use the return value as an output parameter and return a nonnegative value on success. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS

EBADF

fd is not a valid file descriptor.

EFAULT

argp references an inaccessible memory area.

EINVAL

request or *argp* is not valid.

ENOTTY

fd is not associated with a character special device.

ENOTTY

The specified request does not apply to the kind of object that the file descriptor *fd* references.

STANDARDS

No single standard. Arguments, returns, and semantics of **ioctl**() vary according to the device driver in question (the call is used as a catch-all for operations that don’t cleanly fit the UNIX stream I/O model).

The **ioctl**() system call appeared in Version 7 AT&T UNIX.

NOTES

In order to use this call, one needs an open file descriptor. Often the**open**(2) call has unw anted side effects, that can be avoided under Linux by giving it the **O_NONBLOCK** flag.

ioctl structure

Ioctl command values are 32-bit constants. In principle these constants are completely arbitrary, but people have tried to build some structure into them.

The old Linux situation was that of mostly 16-bit constants, where the last byte is a serial number, and the preceding byte(s) give a type indicating the driver. Sometimes the major number was used: 0x03 for the **HDIO_*** ioctls, 0x06 for the **LP*** ioctls. And sometimes one or more ASCII letters were used. For example, **TCGETS** has value 0x00005401, with 0x54 = 'T' indicating the terminal driver, and **CYGETTIMEOUT** has value 0x00435906, with 0x43 0x59 = 'C' 'Y' indicating the cyclades driver.

Later (0.98p5) some more information was built into the number. One has 2 direction bits (00: none, 01:

write, 10: read, 11: read/write) followed by 14 size bits (giving the size of the argument), followed by an 8-bit type (collecting the ioctls in groups for a common purpose or a common driver), and an 8-bit serial number.

The macros describing this structure live in*<asm/ioctl.h>* and are **_IO(type,nr)** and **{_IOR,_IOW,_IOWR}(type,nr,size)**. They use *sizeof(size)* so that size is a misnomer here: this third argument is a data type.

Note that the size bits are very unreliable: in lots of cases they are wrong, either because of buggy macros using *sizeof(sizeof(struct)),* or because of legacy values.

Thus, it seems that the new structure only gave disadvantages: it does not help in checking, but it causes varying values for the various architectures.

SEE ALSO

execve(2), **fcntl**(2), **ioctl_console**(2), **ioctl_fat**(2), **ioctl_ficlone**(2), **ioctl_ficlonerange**(2), **ioctl_fideduperange**(2), **ioctl_fslabel**(2), **ioctl_getfsmap**(2), **ioctl_iflags**(2), **ioctl_ns**(2), **ioctl_tty**(2), **ioctl_userfaultfd**(2), **open**(2), **sd**(4), **tty**(4)