NAME

ddi_copyin – copy data to a driver buffer

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

int ddi_copyin(caddr_t buf, caddr_t driverbuf, size_t cn, int flags);
```

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

buf	Source address from which data is transferred.
driverbuf	Driver destination address to which data is transferred.
cn	Number of bytes transferred.
flags	Set of flag bits that provide address space information about *buf*.

DESCRIPTION

This routine is designed for use in driver **ioctl**(9E) routines for drivers that support layered ioctls. **ddi_copyin( )** copies data from a source address to a driver buffer. The driver developer must ensure that adequate space is allocated for the destination address.

The *flags* argument is used to determine the address space information about *buf*. If the **FKIOCTL** flag is set, this indicates that *buf* is a kernel address, and **ddi_copyin( )** behaves like **bcopy**(9F). Otherwise *buf* is interpreted as a user buffer address, and **ddi_copyin( )** behaves like **copyin**(9F).

Addresses that are word-aligned are moved most efficiently. However, the driver developer is not obliged to ensure alignment. This function automatically finds the most efficient move according to address alignment.

RETURN VALUES

ddi_copyin( ) returns **0**, indicating a successful copy. It returns **–1** if one of the following occurs:

- paging fault; the driver tried to access a page of memory for which it did not have read or write access
- invalid user address, such as a user area or stack area
- invalid address that would have resulted in data being copied into the user block

If a **–1** is returned to the caller, driver entry point routines should return **EFAULT**.

CONTEXT

ddi_copyin( ) can be called from user or kernel context only.

EXAMPLES

A driver **ioctl**(9E) routine (line 12) can be used to get or set device attributes or registers. For the **XX_SETREGS** condition (line 25), the driver copies the user data in **arg** to the device registers. If the specified argument contains an invalid address, an error code is returned.

```
1  struct device  {                        /* layout of physical device registers  */
2    int     control;                      /* physical device control word  */
3    int     status;                       /* physical device status word   */
4    short   recv_char;                    /* receive character from device */
5    short   xmit_char;                    /* transmit character to device  */
6  };

7  struct device_state {
8    volatile struct device *regsp;  /* pointer to device registers */
9    kmutex_t reg_mutex;           /* protect device registers */
     . . .
10 };

11 static void *statep;                     /* for soft state routines */

12 xxioctl(dev_t dev, int cmd, int arg, int mode,
13    cred_t *cred_p, int *rval_p)
14 {
15    struct device_state *sp;
16    volatile struct device *rp;
17    struct device reg_buf;               /* temporary buffer for registers */
18    int instance;

19    instance = getminor(dev);
20    sp = ddi_get_soft_state(statep, instance);
21    if (sp == NULL)
22       return (ENXIO);
23    rp = sp->regsp;
     . . .
24    switch (cmd)  {

25    case XX_SETREGS:                     /* copy data to temp. regs. buf */
26       if (ddi_copyin((caddr_t)arg, (caddr_t)&reg_buf,
27          sizeof (struct device), mode) != 0) {
28             return (EFAULT);
29       }

30       mutex_enter(&sp->reg_mutex);
31       /*
32        * Copy data from temporary device register
33        * buffer to device registers.
34        * e.g. rp->control = reg_buf.control;
35        */
36       mutex_exit(&sp->reg_mutex);
```

```
37          break;
38      }
39  }
```

SEE ALSO

ioctl(9E), **bcopy**(9F), **copyin**(9F), **copyout**(9F), **ddi_copyout**(9F), **uiomove**(9F)

Writing Device Drivers

NOTES

The value of the *flags* argument to **ddi_copyin( )** should be passed through directly from the *mode* argument of **ioctl( )** untranslated.

Driver defined locks should not be held across calls to this function.