NAME

ddi_copyout – copy data from a driver

SYNOPSIS

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

int ddi_copyout(caddr_t *driverbuf***, caddr_t** *buf***, size_t** *cn***, int** *flags***);**

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

driverbuf Source address in the driver from which the data is transferred.

buf Destination address to which the data is transferred.

cn Number of bytes to copy.

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_copyout( )** copies data from a driver buffer to a destination address, *buf*.

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_copyout( )** behaves like **bcopy**(9F). Otherwise *buf* is interpreted as a user buffer address, and **ddi_copyout( )** behaves like **copyout**(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 algorithm according to address alignment.

RETURN VALUES

Under normal conditions a **0** is returned to indicate a successful copy. Otherwise, a **-1** is returned 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_copyout( ) 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. In the **XX_GETREGS** condition (line 25), the driver copies the current device register values to another data area. 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 */
```

```
  short    recv_char;                    /* receive character from device */
  short    xmit_char;                    /* transmit character to device  */
};

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

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

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

   instance = getminor(dev);
   sp = ddi_get_soft_state(statep, instance);
   if (sp == NULL)
      return (ENXIO);
   rp = sp->regsp;
   . . .
   switch (cmd)  {

   case XX_GETREGS:   /* copy registers to arg */
        mutex_enter(&sp->reg_mutex);
        /*
         * Copy data from device registers to
         * temporary device register buffer
         * e.g. reg_buf.control = rp->control;
         */
        mutex_exit(&sp->reg_mutex);

        if (ddi_copyout((caddr_t)&reg_buf, (caddr_t)arg,
           sizeof (struct device), mode) != 0) {
              return (EFAULT);
        }

        break;
   }
}
```

SEE ALSO

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

Writing Device Drivers

NOTES

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

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