NAME

uio, **uiomove**, **uiomove_frombuf**, **uiomove_nofault** - device driver I/O routines

SYNOPSIS

#include <sys/types.h>
#include <sys/uio.h>

```
struct uio {
        struct   iovec *uio_iov;          /* scatter/gather list */
        int      uio_iovcnt;              /* length of scatter/gather list */
        off_t    uio_offset;              /* offset in target object */
        ssize_t  uio_resid;               /* remaining bytes to copy */
        enum     uio_seg uio_segflg; /* address space */
        enum     uio_rw uio_rw;           /* operation */
        struct   thread *uio_td;          /* owner */
};
```

int
uiomove(*void *buf*, *int howmuch*, *struct uio *uiop*);

int
uiomove_frombuf(*void *buf*, *int howmuch*, *struct uio *uiop*);

int
uiomove_nofault(*void *buf*, *int howmuch*, *struct uio *uiop*);

DESCRIPTION

The functions **uiomove**(), **uiomove_frombuf**(), and **uiomove_nofault**() are used to transfer data between buffers and I/O vectors that might possibly cross the user/kernel space boundary.

As a result of any read(2), write(2), readv(2), or writev(2) system call that is being passed to a character-device driver, the appropriate driver *d_read* or *d_write* entry will be called with a pointer to a *struct uio* being passed. The transfer request is encoded in this structure. The driver itself should use **uiomove**() or **uiomove_nofault**() to get at the data in this structure.

The fields in the *uio* structure are:

uio_iov	The array of I/O vectors to be processed. In the case of scatter/gather I/O, this will be more than one vector.
uio_iovcnt	The number of I/O vectors present.

uio_offset	The offset into the device.
uio_resid	The remaining number of bytes to process, updated after transfer.
uio_segflg	One of the following flags:

UIO_USERSPACE	The I/O vector points into a process's address space.
UIO_SYSSPACE	The I/O vector points into the kernel address space.
UIO_NOCOPY	Do not copy, already in object.

uio_rw	The direction of the desired transfer, either UIO_READ or UIO_WRITE.
uio_td	The pointer to a *struct thread* for the associated thread; used if *uio_segflg* indicates that the transfer is to be made from/to a process's address space.

The function **uiomove_nofault**() requires that the buffer and I/O vectors be accessible without incurring a page fault. The source and destination addresses must be physically mapped for read and write access, respectively, and neither the source nor destination addresses may be pageable. Thus, the function **uiomove_nofault**() can be called from contexts where acquiring virtual memory system locks or sleeping are prohibited.

The **uiomove_frombuf**() function is a convenience wrapper around **uiomove**() for drivers that serve data which is wholly contained within an existing buffer in memory. It validates the *uio_offset* and *uio_resid* values against the size of the existing buffer, handling short transfers when the request partially overlaps the buffer. When *uio_offset* is greater than or equal to the buffer size, the result is success with no bytes transferred, effectively signaling EOF.

RETURN VALUES

On success **uiomove**(), **uiomove_frombuf**(), and **uiomove_nofault**() will return 0; on error they will return an appropriate error code.

EXAMPLES

The idea is that the driver maintains a private buffer for its data, and processes the request in chunks of maximal the size of this buffer. Note that the buffer handling below is very simplified and will not work (the buffer pointer is not being advanced in case of a partial read), it is just here to demonstrate the **uio** handling.

```
/* MIN() can be found there: */
```

```
#include <sys/param.h>

#define BUFSIZE 512
static char buffer[BUFSIZE];

static int data_available;      /* amount of data that can be read */

static int
fooread(struct cdev *dev, struct uio *uio, int flag)
{
        int rv, amnt;

        rv = 0;
        while (uio->uio_resid > 0) {
                if (data_available > 0) {
                        amnt = MIN(uio->uio_resid, data_available);
                        rv = uiomove(buffer, amnt, uio);
                        if (rv != 0)
                                break;
                        data_available -= amnt;
                } else
                        tsleep(...);    /* wait for a better time */
        }
        if (rv != 0) {
                /* do error cleanup here */
        }
        return (rv);
}
```

ERRORS

uiomove() and **uiomove_nofault**() will fail and return the following error code if:

[EFAULT] The invoked copyin(9) or copyout(9) returned EFAULT

In addition, **uiomove_nofault**() will fail and return the following error code if:

[EFAULT] A page fault occurs.

SEE ALSO

read(2), readv(2), write(2), writev(2), copyin(9), copyout(9), sleep(9)

HISTORY

The **uio** mechanism appeared in some early version of UNIX.

AUTHORS

This manual page was written by Jörg Wunsch.