
for Motorola Processors
To order single copies of UNIX® SYSTEM V, Release 4 documentation, please call (201) 767-5937.

ATTENTION DOCUMENTATION MANAGERS AND TRAINING DIRECTORS:
For bulk purchases in excess of 30 copies please write to:
Corporate Sales
Prentice Hall
Englewood Cliffs, N.J. 07632
Or call: (201) 461–8441

ATTENTION GOVERNMENT CUSTOMERS: For GSA and other pricing information please call (201) 767-5994.
## Contents

1 **Introduction**
- About This Document 1-1
- Organization of Driver Reference Manuals 1-7
- Conventions Used in This Document 1-8
- Related Learning Materials 1-9

2 **Driver Entry Points (D2)**
- Introduction 2-1
- Overview of Driver Entry-Point Routines and Naming Conventions 2-2
- Manual Pages 2-4

3 **Kernel Functions (D3)**
- Introduction 3-1
- Manual Pages 3-9

4 **Data Structures (D4)**
- Introduction 4-1
- Manual Pages 4-3

A **Appendix A: Error Codes**
- Appendix A: Error Codes A-1
Table of Contents

Appendix B: Migration from Release 3.2 to Release 4.0
Appendix B: Migration from Release 3.2 to Release 4.0 B-1

Index
Index I-1

Permuted Index
Permuted Index 1
Figures and Tables

Figure 1-1: Scope of DDI and DKI 1-1
Table 1-1: Exclusive Entry Points, Functions, and Structures 1-5
Table 1-2: Textual Conventions Used in This Book 1-8
Table 2-1: STREAMS Driver Entry Point Summary 2-2
Table 2-2: Driver Entry Points not Specific to STREAMS 2-3
Table 3-1: STREAMS Kernel Function Summary 3-3
Table 3-2: Kernel Functions Not Specific to STREAMS 3-6
Table 4-1: STREAMS Data Structure Summary 4-1
Table 4-2: Data Structures not Specific to STREAMS 4-2
Table A-1: Driver Error Codes A-2
Table A-2: Error Codes by Driver Routine A-3
Table B-1: 3.2 to 4.0 Migration B-2
# 1 Introduction

## About This Document
- Porting
- Scope of Interfaces
  - Scope of the Device Driver Interface (DDI)
  - Scope of the Driver–Kernel Interface (DKI)
- Interface Members
- Audience
- How to Use This Document

## Organization of Driver Reference Manuals

## Conventions Used in This Document

## Related Learning Materials
- Documentation
  - Driver Development
  - STREAMS
  - C Programming Language and General Programming
  - Assembly Language
  - Operating System
  - Software Packaging
- Training
About This Document

The Device Driver Interface/Driver-Kernel Interface Reference Manual provides reference information needed to write device drivers in the UNIX System V Release 4 environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver-Kernel Interface (DKI). Drivers written to conform to one or both of these interfaces are more likely to be portable to other environments. DDI and DKI address different aspects of the compatibility problem—their differences are summarized in Figure 1-1.

Figure 1-1: Scope of DDI and DKI

<table>
<thead>
<tr>
<th>processor specific routines</th>
<th>DDI only (DxD)</th>
<th>DDI and DKI (DxDK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>supported after SVR4</td>
<td>supported through SVR4</td>
<td></td>
</tr>
</tbody>
</table>

Each box in Figure 1-1 represents a different set of interfaces. The "DDI only" set (indicated throughout this manual with the DxD cross-reference code) are processor specific and are intended to be supported beyond Release 4.0. The DDI described in this manual is specific to the porting base, the 3B2 computer. The "DKI only" set (DxK cross-reference code) are processor independent, but are not guaranteed to be supported in the next release.

Most of the routines, functions, and structures described in this manual are part of both DDI and DKI (cross-referenced by DxDK). As Figure 1-1 shows, drivers written to conform to both interfaces are portable to all AT&T computers supporting UNIX System V Release 4, and they will be compatible through and beyond Release 4. To understand more completely what is meant by "portable" and "compatible" for DDI and DKI, the scope of each interface must be more thoroughly explained.

Introduction 1-1
The goals of DDI and DKI overlap, and are not in any way mutually exclusive. That is, a driver may be written to conform to both interfaces, increasing the chances that driver code can be ported and can remain compatible with future releases of the operating system.

Porting

Software is usually considered portable if it can be adapted to run in a different environment more cheaply than it can be rewritten. The new environment may include a different processor, operating system, and even the language in which the program is written, if a language translator is available. More often, however, software is ported between environments that share an operating system, processor, and source language. The source code is modified to accommodate the differences in compilers or processors or releases of the operating system.

In the past, device drivers did not port easily for one or more of the following reasons:

• To enhance functionality, members had been added to kernel data structures accessed by drivers, or the sizes of existing members had been redefined.

• The calling or return syntax of kernel functions had changed.

• Driver developers did not use existing kernel functions where available, or relied on undocumented side effects that were not maintained in the next release.

• Processor-specific code had been scattered throughout the driver when it could have been isolated.

Operating systems are periodically reissued to customers as a way to improve performance, fix bugs, and add new features. This is probably the most common threat to compatibility encountered by developers responsible for maintaining software. Another common problem is upgrading hardware. As new hardware is developed, customers occasionally decide to upgrade to faster, more capable computers of the same family. Although they may run the same operating system as those being replaced, processor-specific code may prevent the software from porting.
Scope of Interfaces

Although application programs have all of the porting problems mentioned, developers attempting to port device drivers have special challenges. Before describing the differences between DDI and DKI, it is necessary to understand the position of device drivers in UNIX systems.

Device drivers are kernel modules that control data transferred to and received from peripheral devices. Although drivers are configured into a UNIX system as part of the kernel, they are developed independently from the rest of the kernel. If the goal of achieving complete freedom in modifying the kernel is to be reconciled with the goal of binary compatibility with existing drivers, the interaction between drivers and the kernel must be rigorously regulated. This driver/kernel service interface is the most important of the three distinguishable interfaces for a driver, summarized as follows:

- **Driver–Kernel.** I/O System calls result in calls to driver entry point routines. These make up the kernel-to-driver part of the service interface, described in Section 2 of this manual. Drivers may call any of the functions described in Section 3. These are the driver-to-kernel part of the interface.

- **Driver–Hardware.** All drivers (except software drivers) must include an interrupt handling entry point, and may also perform direct-memory access (DMA). These, and other hardware-specific interactions make up the driver/hardware interface.

- **Driver–Boot/Configuration Software.** At boot time, the existence of a driver is made known to the system through information in system files, enabling the system to include the driver. The interaction between the driver and the boot and configuration software is the third interface affecting drivers.

Scope of the Device Driver Interface (DDI)

The primary goal of DDI is to facilitate both source and binary portability across successive releases of UNIX System V on a particular machine. Implicit in this goal is an important fact. Although there is only one DKI, each processor product has its own DDI. Therefore, if a driver is ever to be ported to different hardware, special attention must be paid to the machine-specific routines that make up the “DDI only” part of a driver. These include but are not confined to...
the driver/hardware interface (as described in the previous section). Some processor-specific functionality also may belong to the driver/kernel interface, and may not be easy to locate.

To achieve the goal of source and binary compatibility, the functions, routines, and structures specified in a DDI must be used according to these rules.

- Drivers cannot access system state structure (for example, u and sysinfo) directly.
- For structures external to the driver that may be accessed directly, only the utility functions provided in Section 3 of this manual should be used. More generally, these functions should be used wherever possible.
- The header file ddi.h must be included at the end of the list of header files. This header file "undefines" several macros that are reimplemented as functions.

Scope of the Driver–Kernel Interface (DKI)

As its name implies, the DKI (Driver–Kernel Interface) is a defined service interface for the entry point routines and utility functions specified for communication between the driver and kernel. It does not encompass the driver/hardware or the driver/boot software interface.

Information is exchanged between the driver and kernel in the form of data structures. The DKI specifies the contents of these structures as well as the calling and return syntax of the entry points and utility functions.

The intent of DKI is to promote source portability across implementations of UNIX System V on different machines, and applies only to System V Release 4. Because DKI applies only to the driver/kernel interface, it must be understood that the sections of driver code affecting the hardware and boot/configuration interfaces may need to be rewritten, and should be isolated in subroutines as much as possible.
About This Document

NOTE Certain interfaces documented in the DKI are not part of the DDI. Driver writers should be aware that the use of these interfaces is not guaranteed to be supported beyond System V Release 4.

Interface Members

As noted before, most entry points (Section 2), functions (Section 3), and structures (Section 4) described in this manual belong to both DDI and DKI. Table 1-1 lists the those that are exclusive either to DDI or DKI.

Table 1-1: Exclusive Entry Points, Functions, and Structures

<table>
<thead>
<tr>
<th>Section</th>
<th>DDI only</th>
<th>DKI only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 2</td>
<td>init, int, size, start</td>
<td>segmap, mmap</td>
</tr>
<tr>
<td>Section 3</td>
<td>dma_pageio, etoimajor,</td>
<td>hat_getkpfnnum</td>
</tr>
<tr>
<td></td>
<td>getemajor, geteminor,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getvec, hdeeqd, hdelog,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>itoemajor, kvtophys,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>physiOCK, vtop</td>
<td></td>
</tr>
<tr>
<td>Section 4</td>
<td>hdedata</td>
<td>None</td>
</tr>
</tbody>
</table>

Audience

This manual is for experienced C programmers responsible for creating, modifying, or maintaining drivers that run on AT&T UNIX System V Release 4 and beyond. It assumes that the reader is familiar with UNIX system internals and the advanced capabilities of the C Programming Language. See the “Related Learning Materials” section for a list of available AT&T documents and courses.
How to Use This Document

This manual is organized into four sections and two appendixes:

- "Section 1: Introduction" introduces the DDI, DKI, and other driver interfaces, lists the notational conventions used in this document, and lists related courses and documents.
- "Section 2: Driver Entry Points" contains reference pages for all driver entry point routines.
- "Section 3: Kernel Functions" contains reference pages for all driver functions used in DDI/DKI drivers.
- "Section 4: Data Structures" contains reference pages for structures used in DDI/DKI drivers.
- "Appendix A: Error Codes" contains a list of the error codes that are appropriate for use in DDI/DKI drivers.
- "Appendix B: Migration from Release 3.2 to Release 4.0" describes the changes to DDI/DKI between Release 3.2 and Release 4 of System V.
Organization of Driver Reference Manuals

Driver reference manual pages are similar to those in the *Programmer’s Reference Manual*, with the page name followed by a section number in parentheses. All driver reference manual entries begin with a “D” to distinguish them as driver reference pages.

Currently, the reference pages for the different interfaces are published in separate volumes. Each manual contains three sections:

- **D2** driver entry points
- **D3** kernel functions used by drivers
- **D4** system data structures accessed by drivers

Each section number is suffixed with a letter indicating the interfaces covered. The suffixes used are:

- **D** Device Driver Interface (DDI)
- **K** Driver–Kernel Interface (DKI)
- **DK** DDI and DKI
- **I** SCSI Device Interface (SDI)
- **P** Portable Device Interface (PDI)
- **X** Block and Character Interface (BCI)

For example, `open(D2DK)` refers to the `open` entry point routine for a driver, not to the `open(2)` system call documented in the *Programmer’s Reference Manual*. 
Conventions Used in This Document

Table 1-2 lists the textual conventions used in this book.

Table 1-2: Textual Conventions Used in This Book

<table>
<thead>
<tr>
<th>Item</th>
<th>Style</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Reserved Words</td>
<td>Constant Width</td>
<td>typedef</td>
</tr>
<tr>
<td>C typedef Declarations</td>
<td>Constant Width</td>
<td>caddr_t</td>
</tr>
<tr>
<td>Driver Routines</td>
<td>Constant Width</td>
<td>open routine</td>
</tr>
<tr>
<td>Error Values</td>
<td>Constant Width</td>
<td>EINTR</td>
</tr>
<tr>
<td>File Names</td>
<td>Constant Width</td>
<td>sys/conf.h</td>
</tr>
<tr>
<td>Flag Names</td>
<td>Constant Width</td>
<td>B_WRITE</td>
</tr>
<tr>
<td>Kernel Macros</td>
<td>Constant Width</td>
<td>minor</td>
</tr>
<tr>
<td>Kernel Functions</td>
<td>Constant Width</td>
<td>ttopen</td>
</tr>
<tr>
<td>Kernel Function Arguments</td>
<td><em>Italic</em></td>
<td>bp</td>
</tr>
<tr>
<td>Structure Members</td>
<td>Constant Width</td>
<td>b_addr</td>
</tr>
<tr>
<td>Structure Names</td>
<td>Constant Width</td>
<td>buf structure</td>
</tr>
<tr>
<td>Symbolic Constants</td>
<td>Constant Width</td>
<td>NULL</td>
</tr>
<tr>
<td>System Calls</td>
<td>Constant Width</td>
<td>ioctl(2)</td>
</tr>
<tr>
<td>C Library Calls</td>
<td>Constant Width</td>
<td>printf(3S)</td>
</tr>
<tr>
<td>Shell Commands</td>
<td>Constant Width</td>
<td>layers(1)</td>
</tr>
<tr>
<td>User-Defined Variable</td>
<td><em>Italic</em></td>
<td>prefixclose</td>
</tr>
</tbody>
</table>
Related Learning Materials

AT&T provides a number of documents and courses to support users of our systems. For a listing see:

*AT&T Computer Systems Documentation Catalog (300-000)*
*AT&T Computer Systems Education Catalog (300-002)*

Documentation

Most documents listed here are available from the AT&T Customer Information Center. Refer to the six-digit select code (in parentheses, following the document title) when ordering.

If ordering by telephone, use the following numbers:

1-800-432-6600 (toll free within the continental United States)
1-317-352-8557 (outside the continental United States)

In addition to AT&T documents, the following list includes some commercially available documents that are relevant.

Driver Development


The *UNIX System V and V/386, Release 3, Block and Character Interface (BCI) Driver Reference Manual* (307-192) includes UNIX System V Release 3 reference material to be used in conjunction with the above manual. It describes driver entry point routines (Section D2X), kernel-level functions used in BCI drivers (Section D3X), and data structures accessed by BCI drivers (Section D4X).

The *UNIX System V PDI Driver Design Reference Manual* (305-014) defines the kernel functions and data structures used for Portable Driver Interface (PDI) drivers.

The *UNIX System V SCSI Driver Interface (SDI), Driver Design Reference Manual* (305-009) defines the kernel functions and data structures used for SDI drivers.
STREAMS

The *Programmer's Guide: STREAMS* tells how to write drivers and access devices that use the STREAMS driver interface for character access.

C Programming Language and General Programming


The *Programmer's Guide: Networking Interfaces* provides detailed information, with examples, on the Section 3N library that comprises the UNIX System Transport Level Interface (TLI).

The *Programmer's Guide: ANSI C and Programming Support Tools* includes instructions on using a number of UNIX utilities, including `make` and `SCCS`.

Assembly Language

The *AT&T 3B2/3B5/3B15 Computers Assembly Language Programming Manual* (305-000) describes the Assembly Language instructions used by AT&T 3B2, 3B15 and 3B4000 computers.

*WE 32100 Microprocessor Information Manual, Maxicomputing in Microspace* (307-730) introduces the WE 32100 microprocessor and summarizes its available support products.
Operating System

Bach, Maurice J., *Design of the UNIX Operating System* (320-044), Englewood Cliffs, New Jersey: Prentice-Hall, 1986, discusses the internals of the UNIX operating system, and includes an explanation of how drivers relate to the rest of the kernel.

The UNIX System V reference manuals are the standard reference materials for the UNIX operating system. This information is organized into three books, published separately for each system:

- The *System Administrator's Reference Manual* includes information on administrative commands (Section 1M), special device files (Section 7), and system-specific maintenance commands (Section 8).
- The *Programmer's Reference Manual* includes information on programming commands (Section 1), system calls (Section 2), library routines (Section 3), file formats (Section 4), and miscellaneous topics (Section 5).
- The *User's Reference Manual* includes information on UNIX system user-level commands (Section 1).

Software Packaging

The *Programmer's Guide: System Services and Application Packaging Tools* describes how to write the scripts necessary to install a driver (or other software) under the System Administration utility.

Training

The following courses are of particular interest to driver writers. To register for a class:

- Within the continental United States, call 1-800-TRAINER.
- Within Canada, call 1-800-221-1647.
- Outside the continental United States, call 1-201-953-7554.
Related Learning Materials

*C Language for Experienced Programmers* (UC1001) is a thorough, formal introduction to the C Programming Language.

*Internal UNIX System Calls and Libraries Using C Language* (UC1011) is an introduction to UNIX application programming in C. Topics include the execution environment, memory management, input/output, record and file locking, process generation, and interprocess communication (IPC).

*UNIX System V Release 4 Device Drivers* (UC1056) explores device driver mechanisms, operating system supplied functions, device driver source code examples, installation procedures and debugging techniques. Character, STREAMS, and block devices are covered as well as the entire I/O subsystem.

*UNIX System V Release 4 Internals* (UC1057) presents an in-depth look at UNIX System V, Release 4, including the process, file and I/O subsystems. New UNIX System V Release 4 concepts such as Network File Sharing (NFS), fast file system, and virtual file systems (VFS) are also reviewed.

*Internal System Calls and Libraries (Part 1)* (UC1058) presents the C language programmer's interface to UNIX System V Release 4. This course covers those system calls and library functions not pertaining to interprocess communication. Interprocess communication system calls and library functions are covered in Part 2 of this course.

*Internal System Calls and Libraries (Part 2)* (UC1059) presents UNIX System V Release 4 system calls and library functions pertaining to interprocess communication.
## Table of Contents

**2. Driver Entry Points (D2)**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2-1</td>
</tr>
<tr>
<td>Overview of Driver Entry-Point Routines and Naming Conventions</td>
<td>2-2</td>
</tr>
<tr>
<td>Manual Pages</td>
<td>2-4</td>
</tr>
</tbody>
</table>
Introduction

This chapter describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver. These are called entry-point routines because they provide the calling and return syntax from the kernel into the driver. For all driver types, these routines are called in response to system calls, when the computer is started, when a device generates an interrupt, or for STREAMS drivers, in response to STREAMS events.

All driver routines common to both DDI and DKI are identified with the (D2DK) cross reference code. All DDI-only or DKI-only routines are identified with the (D2D) or (D2K) reference codes respectively.

Functions provided to allow the driver to communicate with the kernel are described in Section 3, and use the (D3DK), (D3D), and (D3K) cross reference codes.

In this section, reference pages contain the following headings:

- **NAME** describes the routine's purpose.
- **SYNOPSIS** summarizes the routine's calling and return syntax.
- **ARGUMENTS** describes each of the routine's arguments.
- **DESCRIPTION** provides general information about the routine.
- **DEPENDENCIES** lists possible dependent routine conditions.
- **SEE ALSO** gives sources for further information.
Overview of Driver Entry-Point Routines and Naming Conventions

Each driver is organized into two parts: the base level and the interrupt level. The base level interacts with the kernel and the user program; the interrupt level interacts with the device.

To uniquely identify a driver, a prefix string is added to the driver routine names. The prefix is defined in the driver's master file. For a driver with the `pre` prefix, the driver code may contain routines named `pre_open`, `pre_close`, `pre_init`, `pre_int`, and so forth. All global variables associated with the driver should also use the same prefix.

System routines can call subroutines that are assigned names by the driver writer. Subroutines should be declared as `static`, and should also use the driver prefix to increase code readability.

Table 2-1 summarizes the STREAMS driver entry points described in this section. These entry points may be used in either DDI or DKI.

**Table 2-1: STREAMS Driver Entry Point Summary**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>put</code></td>
<td>receive messages from the preceding queue</td>
</tr>
<tr>
<td><code>srv</code></td>
<td>service queued messages</td>
</tr>
</tbody>
</table>
Table 2-2 summarizes the block I/O driver entry points described in this section. These entry points may be used in either DDI or DKI, except as noted.

**Table 2-2: Driver Entry Points not Specific to STREAMS**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>chpoll</td>
<td>poll entry point for a non-STREAMS character driver</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>relinquish access to a device</td>
<td></td>
</tr>
<tr>
<td>driverinfo</td>
<td>driver information control</td>
<td>DDI only</td>
</tr>
<tr>
<td>dump</td>
<td>prepare device for crash dump</td>
<td>DDI only</td>
</tr>
<tr>
<td>init</td>
<td>initialize a device</td>
<td>DDI only</td>
</tr>
<tr>
<td>int</td>
<td>process a device interrupt</td>
<td>DDI only</td>
</tr>
<tr>
<td>ioctl</td>
<td>control a character device</td>
<td></td>
</tr>
<tr>
<td>map</td>
<td>map boot arguments to dev_t</td>
<td>DDI only</td>
</tr>
<tr>
<td>mmap</td>
<td>return page frame number</td>
<td>DKI only</td>
</tr>
<tr>
<td>open</td>
<td>gain access to a device</td>
<td></td>
</tr>
<tr>
<td>print</td>
<td>display a driver message on system console</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td>read data from a device</td>
<td></td>
</tr>
<tr>
<td>segmap</td>
<td>map device memory into user space</td>
<td>DKI only</td>
</tr>
<tr>
<td>size</td>
<td>return size of logical device</td>
<td>DDI only</td>
</tr>
<tr>
<td>start</td>
<td>start access to a device</td>
<td>DDI only</td>
</tr>
<tr>
<td>strategy</td>
<td>perform block I/O</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>write data to a device</td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

2. System Calls

intro(D2DK) ................................................................. introduction to driver entry point routines
chpoll(D2DK) ............................................................... poll entry point for a non-STREAMS character driver
close(D2DK) .......................................................................................... relinquish access to a device
driverinfo(D2DK) .......................................................................... communicate with device driver
init(D2D) ................................................................................................................... initialize a device
int(D2D) ..................................................................................................... process a device interrupt
ioctl (D2DK) ................................................................................................ control a character device
mapdevice(D2DK) ............................................................... map boot information into root dev_t
mmap(D2K) .......................................................................................... check virtual mapping for memory mapped device
open (D2DK) .................................................................................................... gain access to a device
print(D2DK) ............................................................... display a driver message on system console
put(D2DK) .......................................................................................... receive messages from the preceding queue
read(D2DK) ................................................................................................... read data from a device
segmap(D2K) .......................................................................................... map device memory into user space
size(D2D) ................................................................................................ return size of logical device
srv(D2DK) ................................................................................................... service queued messages
start(D2D) ........................................................................................................ start access to a device
strategy(D2DK) .......................................................................................... perform block I/O
write(D2DK) ..................................................................................................... write data to a device
NAME
  intro - introduction to driver entry point routines

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

DESCRIPTION
  This section describes the routines a developer needs to include in a device
driver. These routines are called "entry point routines" because they provide the
interfaces that the kernel needs from drivers. The kernel calls them when needed.
Some are called at well-defined times, such as system start up and system shut
down. Others are called as a result of I/O-related system calls or external events,
such as interrupts from peripheral devices.

  Each driver is organized into two logical parts: the base level and the interrupt
level. The base level interacts with the kernel and the device on behalf of
processes performing I/O operations. The interrupt level interacts with the
device and the kernel as a result of an event such as data arrival, and usually can­
not be associated with any particular process.

  Each driver is uniquely identified by a prefix string specified in its configuration
file. The name of all the driver-supplied routines and global variables should
begin with this prefix. This will reduce the chance of a symbol collision with
another driver. Any private routines defined by a driver that are not entry point
routines should be declared as static. Also, any global variables that are
private to the driver should be declared as static.
NAME
chpoll - poll entry point for a non-STREAMS character driver

SYNOPSIS
#include <sys/poll.h>

chpoll(dev_t dev, short events, int anyyet, short *reventsp,
       struct pollhead **phpp);

ARGUMENTS

dev   The device number for the device to be polled.

events The events that may occur. Valid events are:
        POLLIN  Data are available to be read.
        POLLOUT Data may be written without blocking.
        POLLPRI High priority data may be read.
        POLLHUP A device hangup.
        POLLERR A device error.

anyyet A flag that is non-zero if any other file descriptors in the pollfd array
        have events pending. The poll(2) system call takes a pointer to an
        array of pollfd structures as one of its arguments. See the poll(2)
        reference page for more details.

reventsp A pointer to a bitmask of the returned events satisfied.

phpp   A pointer to a pointer to a pollhead structure. The pollhead struc-
        ture is defined in sys/poll.h.

DESCRIPTION
The chpoll entry point routine is used by non-STREAMS character device
drivers that wish to support polling. The driver must implement the polling
discipline itself. The following rules must be followed when implementing the
polling discipline:
1. Implement the following algorithm when the chpoll entry point is called:
   if (events_are_satisfied_now) {
       *reventsp = mask_of_satisfied_events;
   } else {
       *reventsp = 0;
       if (!anyyet)
               *phpp = &my_local_pollhead_structure;
   }
   return (0);

2. Allocate an instance of the pollhead structure. This instance may be tied to
   the per-minor data structure defined by the driver. The pollhead structure
   should be treated as a "black box" by the driver. None of its fields should
   be referenced. However, the size of this structure is guaranteed to remain
   the same across releases.

3. Call the pollwakeup(D3DK) function whenever an event of type events
   listed above occur. This function should only be called with one event at a
time.
RETURN
A chpoll routine should return 0 for success, or the appropriate error number.

SEE ALSO
pollwakeup(D3DK), poll(2)
NAME

close - relinquish access to a device

SYNOPSIS  [Block and Character]

#include <sys/types.h>
#include <sys/file.h>
#include <sys/errno.h>
#include <sys/open.h>
#include <sys/cred.h>
#include <sys/ddi.h>

int prefixclose(dev_t dev, int flag, int otyp, cred_t *cred_p);

ARGUMENTS

dev  Device number.

flag  File status flag, as set by the open(2) or modified by the fcnt1(2) sys­
tem calls. The flag is for information only—the file should always be
closed completely. The flag is taken from the f_flag member of the
file structure which is in file.h. Possible values are: FEXCL,
FNDELAY, FREAD, and FWRITE. Refer to open(D2D) for more informa­
tion.

otyp Parameter supplied so that the driver can determine how many times
a device was opened and for what reasons. The flags assume the open
routine may be called many times, but the close routine should only
be called on the last close of a device.

OTYP_BLK close was through block interface for the device
OTYP_CHAR close was through the raw/character interface for the
device
OTYP_MNT close was called as a result of a umount(2) system call;
unmount the file system associated with the block
device
OTYP_SWP close a swapping device
OTYP_LYR close a layered process (a higher-level driver called the
close routine of the device)

OTYP_DMP close a device previously opened for crash dump

*cred_p  Pointer to the cred(D4D) user credential structure.

SYNOPSIS  [STREAMS]

#include sys/types.h
#include sys/stream.h
#include sys/file.h
#include sys/errno.h
#include sys/open.h
#include sys/cred.h
#include sys/ddi.h

int prefixclose(queue_t *q, int flag, cred_t *cred_p);
ARGUMENTS

*q Pointer to queue structure used to reference the read side of the driver. (A queue is the central node of a collection of structures and routines pointed to by a queue.)

flag File status flag.

*cred_p Pointer to the cred(D4DK) user credential structure.

DESCRIPTION

For STREAMS drivers, the close routine is called by the kernel through the cdevsw table entry for the device. (Modules use the fmodsw table.) A non-null value in the d_str field of the cdevsw entry points to a streamtab structure, which points to a qinit structure containing a pointer to the close routine. Non-STREAMS close routines are called directly from the bdevsw (block) or cdevsw (character) tables.

The close routine ends the connection between the user process and the device, and prepares the device (hardware and software) so that it is ready to be opened again.

A device may be opened simultaneously by multiple processes and the open driver routine is called for each open, but the kernel will only call the close routine when the last process using the device issues a close(2) or umount(2) system call or exits. (An exception is a close occurring with the otyp argument set to OTYP_LYR, for which a close (also having otyp = OTYP_LYR) occurs for each open.)

In general, a close routine should always check the validity of the minor number component of the dev parameter. The routine should also check permissions as necessary, by using the cred(D4DK) structure (if pertinent), and the appropriateness of the flag and otyp parameter values.

A close routine could perform any of the following general functions:

- disable interrupts
- hang up phone lines
- rewind a tape
- deallocate buffers from a private buffering scheme
- unlock an unsharable device (that was locked in the open routine)
- flush buffers
- notify a device of the close
- deallocate any resources allocated on open

The close routines of STREAMS drivers and modules are called when a stream is dismantled or a module popped. The steps for dismantling a stream are performed in the following order. First, any multiplexor links present are unlinked and the lower streams are closed. Next, the following steps are performed for each module or driver on the stream, starting at the head and working toward the tail:

1. The write queue is given a chance to drain.
2. The close routine is called.
3. The module or driver is removed from the stream.
RETURN VALUE
The close routine should return 0 for success, or the appropriate error number. Refer to Appendix A for a list of DDI/DKI error numbers. Return errors rarely occur, but if a failure is detected, the driver should decide whether the severity of the problem warrants either displaying a message on the console or, in worst cases, triggering a system panic. Generally, a failure in a close routine occurs because a problem occurred in the associated device.

SEE ALSO
open(D2D), cred(D4DK)
NAME
driverinfo - communicate with device driver

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/dinfo.h>
#include <sys/edt.h>

int prefixdriverinfo(dev, cmd, arg1, arg2);

ARGUMENTS
dev the dev_t for the device on which the information exchange is to take
     place.

   cmd the command to be performed (int)
   arg1 a general argument (int)
   arg2 a general argument (int)

DESCRIPTION
The driverinfo routine is called indirectly (through cdevsw) by the kernel to
communicate with a device driver without having to open a device first (via open
and ioctl).

Drivers are free to support whatever commands they see fit (for example,
firmware download or specialized hardware controller commands), and return
whatever error codes seem appropriate, but drivers that implement the standard­
ized driverinfo commands listed below must adhere to the error code returns
specified by the standardized command description.

STANDARDIZED COMMAND DXGETEDT
This command returns the extended EDT table for the devices attached to the
controller for the device specified by the dev argument. The arg1 argument is the
user space virtual address of the place to return the information, and the arg2
argument is the byte count.

The extended EDT table is retrieved with two calls via driverinfo. First, the
number of extended EDT entries for the controller specified by the device argu­
ment is obtained by calling sysm88k with a buffer of one integer and a byte count
of sizeof(int). This integer (that is, the number of extended EDT structures for the
argument controller) is then used to calculate how large a buffer is needed to
contain the entire extended EDT table for the controller, and that buffer is then
obtained. The size of this buffer must be sizeof(int) + XEDT_count * sizeof(struct
xedt). The second invocation of driverinfo with this newly obtained buffer
retrieves the extended EDT table.

If the number of bytes specified by arg2 is incorrect, EINVAL is return. If the
controller specified by the dev argument isn't present, ENXIO is returned.
STANDARDIZED COMMAND DDEVXEDT
This command returns the extended EDT entry for the device specified by the `dev` argument. The `arg1` argument is the user space virtual address of an xedt structure which will be filled in by this operation.

If the device specified by the `dev` argument isn’t present, ENXIO is returned.

GENERIC ERROR RETURNS
If the command is invalid, EINVAL is returned.

Any I/O error incurred by any operation will return EIO.

If a copyin or copyout operation fails, EFAULT is returned.

SEE ALSO
ioctl(D2DK)
NAME
init - initialize a device

SYNOPSIS
void prefixinit();

DESCRIPTION
init and start(D2D) routines are used to initialize drivers and the devices they control. init routines are executed during system initialization, and can be used in drivers that do not require low level system services in order to be initialized. start routines are executed after low level services are enabled, such as interrupts and lower level kernel interfaces, but before file systems are available. Most drivers can use either an init or a start routine, or they can be used in combination. However, an init routine must be used in any driver controlling a device required to bring the system up.

Not all drivers need an init or a start routine. However, a driver must have either an init or start routine if it needs to allocate any data structures.

init and start routines can perform functions such as:
- allocating buffers for private buffering schemes
- mapping a device into virtual address space
- initializing hardware (for example, system generation or resetting the board)
- initializing a serial device in a character driver

Because the init and start routines are executed before there is user context, no functions that require user-context, such as sleep(D3DK), may be called.

SEE ALSO
start(D2D), cunix(1M), master(4)
NAME
int - process a device interrupt

SYNOPSIS
void prefixint(int ctlr);

ARGUMENT
ctlr specifies the unique number associated with the controller that is generating the interrupt. These numbers are unique within each type of controller and start with zero. ctlr represents the controller number extracted from the board field of the /stand/.edt_data file.

DESCRIPTION
The int routine is the interrupt handler for both block and character hardware drivers. The interrupt handler is responsible for determining the reason for an interrupt, servicing the interrupt, and waking up any base-level driver processes sleeping on the interrupt completion. For example, when a disk drive has transferred information to the host to satisfy a read request, the disk drive’s controller generates an interrupt. The CPU acknowledges the interrupt and calls the interrupt handler associated with that controller and disk drive. The interrupt routine services the interrupt and then wakes up the driver base-level process waiting for data. The base-level portion of the driver then conveys the data to the user.

In general, most interrupt routines must do the following tasks:
- keep a record of interrupt occurrences
- return immediately if no devices controlled by a driver caused the interrupt (only for systems supporting shared interrupts)
- interpret the interrupt routine argument ctlr
- reject requests for devices that are not served by the device’s controller
- process interrupts that happen without cause (called spurious interrupts)
- handle all possible device errors
- wake processes that are sleeping on the resolution of an interrupt request

There are also many tasks the int routine must perform that are driver-type and device specific. For example, the following types of drivers require different functions from their int routines:

- A block driver dequeues requests, wakes up processes sleeping on an I/O request, and ensures that system generation has completed.
- A terminal driver receives and sends characters.
- A printer driver ensures that characters are sent.

In addition, the functions of an int routine are device dependent. You should know the exact chip set that produces the interrupt for your device. You need to know the exact bit patterns of the device’s control and status register and how data is transmitted into and out of your computer. These specifics differ for every device you access.
The int routine for an intelligent controller that does not use individual interrupt vectors for each subdevice must access the completion queue to determine which subdevice generated the interrupt. It must also update the status information, set/clear flags, set/clear error indicators, and so forth to complete the handling of a job. The code should also be able to handle a spurious completion interrupt identified by an empty completion queue. When the routine finishes, it should advance the unload pointer to the next entry in the completion queue.

If the driver called biowait(3DK) or sleep(3DK) to await the completion of an operation, the int routine must call biodone(3DK) or wakeup(3DK) to signal the process to resume.

int is only used with hardware drivers, not software drivers.

CAUTION: The int routine must never:

- contain calls to the sleep kernel function
- use functions that call sleep
- drop the interrupt priority level below the level at which the interrupt routine was entered
- call any function or routine that requires user context (that is, if it accesses or alters information associated with the running process)

Note: uio_move(3DK) cannot be used in an interrupt routine when the uio_segflg member of the uio(4DK) structure is set to UIO USERSPACE (indicating a transfer between user and kernel space).

SEE ALSO

biowait(3DK), sleep(3DK), biodone(3DK), wakeup(3DK)
NAME
ioctl - control a character device

SYNOPSIS
#include <sys/cred.h>
#include <sys/types.h>
#include <sys/errno.h>

int prefixioctl(dev_t dev, int cmd, int arg, int mode, cred_t *cred_p,
   int *rval_p);

ARGUMENTS
   dev       Device number.
   cmd       Command argument the driver ioctl routine interprets as the operation to be performed. It should be defined, along with an integer value that is actually passed, in the header file.

   The I/O control command name and value can be defined in the driver code itself, but this is not recommended. If I/O control commands are defined in a header file, the user program and the driver can both access the same definitions to ensure that they agree about what each I/O control command value represents.

   The I/O control command name is traditionally an all uppercase alphabetic string. This alphabetic name can be a mnemonic. You should try to keep the values for your I/O control commands distinct from others on the system. Each driver's I/O control commands are discrete, but it is possible for user-level code to access a driver with an I/O control command that is intended for another driver, which can lead to serious consequences, such as if it meant to pass "drop carrier on a communication line," but instead sends the argument to a disk where it is interpreted as "reformat drive." Permissions can be set to prevent most such events, but the more unique your I/O control command values are, the safer you are.

   A number of different schemes are legal for assigning values to I/O control command names. The most straightforward is to use decimal numbers; for example
   
   #define COMMAND101
   #define COMMAND202

   Similarly, one can assign hexadecimal numbers as values

   #define COMMANDA 0x0a
   #define COMMANDFF 0xff

   The drawback to these methods is that one quickly gets an operating system that contains several instances of each I/O control command value, with the inherent risks discussed above.

   A common method to assign I/O control command values that are less apt to be duplicated is to use a left-shifted 8 scheme. For instance
ioctl (D2DK)  

Alternately, the shift-left-8 scheme can be defined as a constant then used for the I/O control command definitions. For example

```c
#define COMMAND10 ('Q'<<8|10)
#define COMMAND11 ('Q'<<8|11)
#define COMMAND12 ('Q'<<8|12)
```

An alternative coding style is to use enumerations for the command argument, which allows the compiler to do additional type checking.

```c
typedef enum {
    XX_COMMAND10 = 'Q'<<8 | 10,
    XX_COMMAND11 = 'Q'<<8 | 11,
    XX_COMMAND12 = 'Q'<<8 | 12,
} xx_cmds_t;
```

termio(7) specifies the command types that must work for AT&T terminal drivers. Terminal drivers typically have a command to read the current ioctl settings and at least one other that defines new settings.

**arg**

Passes parameters between a user program and the driver.

When used with terminals, the argument is the address of a user program structure containing driver or hardware settings. Alternatively, the argument may be an integer that has meaning only to the driver. The interpretation of the argument is driver dependent and usually depends on the command type; the kernel does not interpret the argument.

**mode**

Contains values set when the device was opened.

Use of this mode is optional. However, the driver may use it to determine if the device was opened for reading or writing. The driver makes this determination by checking the FREAD or FWRITE setting (values are in file.h).

See the flag argument description of the open routine for further values for the ioctl routine's mode argument.

**cred_p**

Pointer to the cred(D4DK) user credential structure.

**rval_p**

Pointer to return value for calling process. The driver may elect to set the value which is valid only if the ioctl(D2DK) succeeds.

**DESCRIPTION**

The ioctl(D2DK) routine provides character-access drivers with an alternate entry point that can be used for almost any operation other than a simple transfer of characters in and out of buffers. Most often, ioctl is used to control device hardware parameters and establish the protocol used by the driver in processing data.
The kernel looks up the device’s file table entry, determines that this is a character device, and looks up the entry point routines in cdevsw. The kernel then packages the user request and arguments as integers and passes them to the driver’s ioctl routine. The kernel itself does no processing of the passed command, so it is up to the user program and the driver to agree on what the arguments mean.

I/O control commands are used to implement the terminal settings passed from ttymon(1M) and stty(1), to format disk devices, to implement a trace driver for debugging, and to clean up character queues. Since the kernel does not interpret the command type that defines the operation, a driver is free to define its own commands.

Drivers that use an ioctl routine typically have a command to “read” the current ioctl settings, and at least one other that sets new settings. You can use the mode argument to determine if the device unit was opened for reading or writing, if necessary, by checking the FREAD or FWRITE setting.

If the third argument, arg, is a pointer to user space, the driver should call the copyin(D3DK) and copyout(D3DK) functions to transfer data between kernel and user space.

To implement I/O control commands for a driver the following two steps are required:

1. Define the I/O control command names and the associated value in the driver’s header file and comment the commands.
2. Code the ioctl routine in the driver that defines the functionality for each I/O control command name that is in the header file.

The ioctl routine is coded with instructions on the proper action to take for each command. It is basically a switch statement, with each case definition corresponding to an ioctl name to identify the action that should be taken. However, the command passed to the driver by the user process is an integer value associated with the command name in the header file.

It is critical that command definitions and routines be clearly commented. Because there is so much flexibility in how commands are used, uncommented commands can be very difficult to interpret at a later time.

Terminal drivers use and support the ioctl commands defined on the termio(7) manual page. For instance, TGETA gets the parameters associated with the terminal and stores them in the structure referenced in the third argument of the routine call. TCSETA sets the parameters associated with the terminal from the structure referenced in the third argument.

Note: STREAMS drivers do not have ioctl routines. The stream head converts I/O control commands to M_IOCTL messages, which are handled by the driver’s put(D2DK) or srv(D2DK) routine.

**RETURN VALUE**

The ioctl routine should return 0 for success, or the appropriate error number. Refer to Appendix A for a list of DDI/DKI error numbers. The driver may also set the value returned to the calling process through the rval_p pointer.
SEE ALSO
   copyin(D3DK), copyout(D3DK)
NAME
mapdevice - map boot information into root dev_t

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

int prefixmapdevice(struct bootcmd *info, dev_t dev);

ARGUMENTS
info Pointer to the bootcmd structure.
dev Pointer to a dev_t.

DESCRIPTION
The mapdevice routine is called indirectly (through cdevsw) by the kernel to convert the boot arguments into the dev_t for the boot device. This entry point is optional, but any device driver that may be responsible for the boot device must include it.

At boot time, the kernel offers the bootcmd structure returned by the boot block to each device driver that has a mapdevice entry point. The first device driver to accept the bootcmd information is assumed to have validly mapped the information into the dev_t for the boot device; searching is then discontinued and kernel initialization continues.

For disks, dev_t should represent the slice for the root disk as specified in the bootcmd structure. For tapes, dev_t should represent the no rewind device for a tape at the specified address.

RETURN VALUE
If the device driver is not the device responsible for the boot device or if the device driver is not configured to deal with the device specified by the bootcmd structure, a -1 is returned. Otherwise a zero is returned, and the dev_t corresponding to the bootcmd information is stored.
NAME

mmap - check virtual mapping for memory mapped device

SYNOPSIS

#include <sys/types.h>
#include <sys/cred.h>
#include <sys/nunan.h>
#include <sys/vm.h>

int prefixmmap(dev_t dev, off_t off, int prot);

ARGUMENTS

dev  Device whose memory is to be mapped.
off  Offset within device memory at which mapping begins.
prot Protection flag from nunan.h (for example, PROT_WRITE, PROT_READ).

DESCRIPTION

The mmap entry point is a required entry point for character drivers supporting
memory-mapped devices. A memory mapped device has memory that can be
mapped into a process’s address space. The mmap(2) system call, when applied to
a character special file, allows this device memory to be mapped into user space
for direct access by the user application (no kernel buffering overhead is
required).

An mmap(D2K) routine checks if each offset is within the range of pages sup-
ported by the device. For example, a device that has 512 bytes of memory that
can be mapped into user space should not support offsets greater than 512. If the
offset does not exist, then -1 is returned. If the offset does exist, mmap returns the
masked page table entry for the page at offset off in the device’s memory.

mmap should only be supported for memory-mapped devices or pseudo-devices.
See the segmap(D2K) reference page for further information on memory mapped
device drivers.

RETURN VALUE

If the protection and offset are valid for the device, the driver should return the
masked page table entry, typically obtained using the function
hat_getkpfnum(D3K), for the page at offset off in the device’s memory. If not, -1
should be returned.

SEE ALSO

segmap(D2K), hat_getkpfnum(D3K)
NAME

open - gain access to a device

SYNOPSIS  [Block and Character]

#include <sys/types.h>
#include <sys/file.h>
#include <sys/errno.h>
#include <sys/open.h>
#include <sys/cred.h>

prefixopen(dev_t *dev, int flag, int otyp, cred_t *cred_p);

ARGUMENTS

dev  Pointer to a device number.

flag Information passed from the user program open(2) or create(2) system call instructs the driver on how to open the file. The bit settings for the flag are found in file.h associated with the f_flag member of the file structure. Valid settings are:

FNDELAY open the device and return immediately without sleeping (do not block the open even if there is a problem)

FREAD open the device with read-only permission (if ORed with FWRITE, then allow both read and write access)

FWRITE open a device with write-only permission (if ORed with FREAD, then allow both read and write access)

otyp Parameter supplied so that the driver can determine how many times a device was opened and for what reasons. The flags assume the open routine may be called many times, but the close routine should only be called on the last close of a device. All flags are defined in open.h.

OTYP_BLK open occurred through block interface for the device

OTYP_CHAR open occurred through the raw/character interface for the device

OTYP_MNT the file system on the block device is being opened due to a mount(2) system call

OTYP_DMP open a device for crash dump

OTYP_SWP open a swapping device

OTYP_LVR open a layered process. This flag is used when one driver calls another driver's open or close routine. In this case, there is exactly one close for each open called. This permits software drivers to exist above hardware drivers and removes any ambiguity from the hardware driver regarding how a device is used. This flag applies to both block and character devices.

*cred_p Pointer to the cred(D4DK) user credential structure.
SYNOPSIS  [STREAMS]
#include <sys/file.h>
#include <sys/stream.h>

prefixopen(queue_t *q, dev_t *dev, int oflag, int sflag, cred_t *cred_p);

ARGUMENTS  [STREAMS]
*q  A pointer to the read queue. (A queue is the central node of a collection of structures and routines pointed to by a queue.)
*dev  Pointer to a device number. For modules, *dev always points to the device number associated with the driver at the end (tail) of the stream.
oflag  Valid oflag values are the same as those listed above, with the exception that FAPPEND, FCREAT, and FTRUNC have no meaning to a STREAMS device. For modules, oflag is always set to 0.
sflag  Valid values are as follows:
  CLONEOPEN  Eliminates the need for user processes to poll many minor devices when looking for an unused one. If the driver wishes to assign the device a device file, the open routine must assign and return a minor number. If no device file is required, the open routine does not have to return a minor number.
  MODOPEN  Indicates that an open routine is being called for a module, not a driver. Drivers should return error numbers or 0 if an open is attempted with sflag set to MODOPEN.
  0  Indicates a driver opened directly, without calling the clone driver.
*cred_p  Pointer to the cred(D4DK) user credential structure.

DESCRIPTION
The driver's open routine is called by the kernel through the cdevsw or bdevs  

entry for the device during an open(2) or a mount(2) on the special file for the device. The routine should verify that the minor number component of dev is valid, that the type of access requested by otyp and flag is appropriate for the device, and, if required, check permissions using the user credentials pointed to by cred_p.

RETURN VALUE
The open routine should return 0 for success, or the appropriate error number. Refer to Appendix A for a list of DDI/DKI error numbers.

SEE ALSO
close(D2DK)
NAME
print - display a driver message on system console

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/errno.h>

int prefixprint(dev_t dev, char *str);
```

ARGUMENTS

dev Device number.

*str Pointer to a character string describing the problem. An explanation of the problem contained in the string should be included in the driver output.

DESCRIPTION
The print routine is called indirectly by the kernel through the bdevsw entry for the device when the kernel has detected an exceptional condition (such as out of space) in the device. To display the message on the console, the driver should use the cmn_err(D3DK) kernel function.

RETURN VALUE
The print routine should return 0 for success, or the appropriate error number. Refer to Appendix A for a list of DDI/DKI error numbers. The print routine can fail if the driver implemented a non-standard print routine that attempted to perform error logging, but was unable to complete the logging for whatever reason. Generally, since most print routines call the cmn_err(D3DK) function, and this function is declared as void, return values are seldom returned from this routine. If a failure occurs, call cmn_err to display a message to the operator.

SEE ALSO

cmn_err(D3DK)
NAME
put - receive messages from the preceding queue

SYNOPSIS
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>
void prefixrput(queue_t *q, mblk_t *mp); /* read side */
void prefixwput(queue_t *q, mblk_t *mp); /* write side */

ARGUMENTS
*q Pointer to the queue structure.
mp Pointer to the message block.

DESCRIPTION
The primary task of the put routine is to coordinate the passing of messages from one queue to the next in a stream. The put routine is called by the preceding stream component (module, driver, or stream head). put routines are designated “write” or “read” depending on the direction of message flow.

With few exceptions, a module or driver must have a put routine. One exception is the read side of a driver, which does not need a put routine because there is no component downstream to call it. The put routine is always called before the component’s corresponding srv (service) routine, and so put should be used for the immediate processing of messages.

A put routine must do at least one of the following when it receives a message:
- pass the message to the next component on the stream by calling the putnext function
- process the message, if immediate processing is required (for example, high priority messages)
- enqueue the message (with the putq function) for deferred processing by the service srv routine

Typically, a put routine will switch on message type, which is contained in the db_type member of the datab structure pointed to by mp. The action taken by the put routine depends on the message type. For example, a put routine might process high priority messages, enqueue normal messages, and handle an unrecognized message by changing its type to M_IOCNAK (negative acknowledgement) and sending it back to the stream head using the qreply function.

The putq function can be used as a module’s put routine when no special processing is required and all messages are to be enqueued for the srv routine.

put routines do not have user context and so may not call sleep.

SEE ALSO
The BCI Driver Development Guide, Chapter 7, “STREAMS”
The Programmer’s Guide: STREAMS
streamtab, putctl, putctll, putnext, putq, qreply, srv
NAME
read – read data from a device

SYNOPSIS
#include <sys/types.h>
#include <sys/errno.h>
#include <sys/open.h>
#include <sys/uio.h>
#include <sys/cred.h>

prefixread(dev_t dev, uio *uio_p, cred_t *cred_p);

ARGUMENTS
dev
Device number.

*uio_p Pointer to the uio(D4DK) structure that describes where the data is to
be stored in user space.

*cred_p Pointer to the cred(D4DK) user credential structure for the I/O trans-
action.

DESCRIPTION
The driver read routine is called indirectly through cdevsw by the read(2) sys-
tem call. The read routine should check the validity of the minor number com-
ponent of dev and the user credentials contained in the cred(D4DK) structure
pointed to by *cred_p (if pertinent). The read routine should supervise the data
transfer into the user space described by the uio(D4DK) structure.

RETURN VALUE
The read routine should return 0 for success, or the appropriate error number.
Refer to Appendix A for a list of error values.

SEE ALSO
write(D2DK)
NAME

segmap – map device memory into user space

SYNOPSIS

#include <sys/types.h>
#include <sys/nunan.h>
#include <sys/param.h>
#include <sys/vm.h>

int prefixsegmap(dev_t dev, off_t off, struct as *asp, addr_t *addrp,
off_t len, unsigned int prot, unsigned int maxprot,
unsigned int flags, cred_t *cred_p);

ARGUMENTS

dev Device whose memory is to be mapped.
off Offset within device memory at which mapping begins.
*asp Pointer to the address space into which the device memory should be
mapped.
*addrp Pointer to the address in the address space to which the device
memory should be mapped.
len Length (in bytes) of the memory to be mapped.
prot Protection flag (from sys/nunan.h) for example, PROT_WRITE,
PROT_READ, PROT_USER (indicating the mapping is being done as a
result of a nunap(2) system call).
maxprot Maximum protection flag possible for attempted map (PROT_WRITE
may be masked out if the user opened the special file read-only). If
(maxprot & prot) != prot then there is an access violation.
flags Flags indicating type of mmap (for example, MAP_SHARED vs.
MAP_PRIVATE), whether the user specified an address (MAP_FIXED).
Found in sys/mman.h.
*cred_p Pointer to the cred(D4DK) user credentials structure.

DESCRIPTION

The segmap entry point is an optional routine for character drivers that support
memory mapping. The mmap(2) system call, when applied to a character special
file, allows device memory to be mapped into user space for direct access by the
user application (no kernel buffering overhead is required).

Typically, a character driver that needs to support the mmap(2) system call sup­
plies either a single mmap(D2K) entry point, or both an mmap and a segmap entry
point routine (see the mmap(D2K) reference page). If no segmap entry point is
provided for the driver, the default kernel segmap routine is called to perform the
mapping.

A driver for a memory-mapped device would provide a segmap entry point if it:
requires the mapping to be done through a virtual memory (VM) segment
driver other than the default seg_dev driver provided by the kernel
needs to control the selection of the user address at which the mapping
occurs in the case where the user did not specify an address in the
system call

Among the responsibilities of a segmap entry point are:

Select a segment driver and check the memory map flags for appropriateness to the segment driver. For example, the seg_dev segment driver
does not support memory maps that are marked MAP_PRIVATE (copy-on-write).
Verify that the range to be mapped makes sense in the context of the device (does the offset and length make sense for the device memory that
is to be mapped). Typically, this task is performed by calling the
entry point.
If MAP_FIXED is not set in flags, obtain a user address at which to map. Otherwise, unmap any existing mappings at the user address specified.
Perform the mapping and return the error status if it fails.

RETURN VALUE
The routine returns 0 if the driver is successful in performing the memory map of
its device address space into the specified address space. An error number
should be returned on failure. For example, valid error numbers would be ENXIO
if the offset/length pair specified exceeds the limits of the device memory, or
EINVAL if the driver detects an invalid type of mapping attempted.

SEE ALSO
mmap(D2K)
NAME
size - return size of logical device

SYNOPSIS
#include <sys/types.h>

prefixsize(dev_t dev);

ARGUMENT
dev The logical device number.

DESCRIPTION
Returns the number of 512-byte units on a logical device (partition). Although this routine is not required, it is recommended that new drivers include one as the Release 4.0 kernel calls the size routine on behalf of certain UNIX commands such as stat(3G).

RETURN VALUE
The number of 512 byte units on the logical device specified by dev, or -1 on failure.
NAME

`srv` - service queued messages

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>

void prefixsrv(queue_t q); /* read side */

void prefixwsrv(queue_t q); /* write side */
```

ARGUMENTS

- `*q` Pointer to the queue structure

DESCRIPTION

The optional service (`srv`) routine may be included in a STREAMS module or
driver for one or more of the following reasons:

- to provide greater control over the flow of messages in a stream
- to make it possible to defer the processing of some messages to avoid
  depleting system resources
- to combine small messages into larger ones, or break large messages into
  smaller ones
- to recover from resource allocation failure. A module’s or driver’s
  put routine can test for the availability of a resource, and if it is
  not available, enqueue the message for later processing by the `srv` routine.

A message is first passed to a module’s or driver’s put routine, which
may or may not do some processing. It must then either

- pass the message to the next stream component with putnext function
- if a `srv` routine has been included, it may call the putq function to
  place the message on the queue

Once a message has been enqueued, the STREAMS scheduler controls the calling
of the service routine. Service routines are called in FIFO order by the scheduler.
No guarantees can be made about how long it will take for a `srv` routine to be
called except that it will happen before any user level process are run.

Every stream component (stream head, module or driver) has limit values it uses
to implement flow control. Tunable high and low water marks are checked to
stop and restart the flow of message processing. Flow control limits apply only
between two adjacent components with `srv` routines.

STREAMS messages can be defined to have up to 256 different priorities to sup­
port some networking protocol requirements for multiple bands of data flow. At
a minimum, a stream must distinguish between normal (priority zero) messages
and high priority messages (such as M_IOACK). High priority messages are
always placed at the head of the `srv` routine’s queue, after any other enqueued
high priority messages. Next are messages from all included priority bands,
which are enqueued in decreasing order of priority. Each priority band has its
own flow control limits. If a flow controlled band is stopped, all lower priority
bands are also stopped.

Once a srv routine is called by the STREAMS scheduler it must process all mes­
theses on its queue. The following steps are general guidelines for processing
messages. Keep in mind that many of the details of how a srv routine should be
written depend of the implementation, the direction of flow (upstream or down­
stream), and whether it is for a module or a driver.

1. Use the getq(D3DK) function to get the next enqueued message.
2. If the message is high priority, process (if appropriate) and pass to the next
stream component with the putnext(D3DK) function.
3. If it is not a high priority message (and therefore subject to flow control),
attempt to send it to the next stream component with a srv routine. Use
bcanput(D3DK) to determine if this can be done.
4. If the message cannot be passed, put it back on the queue with
putbq(D3DK). If it can be passed, process (if appropriate) and pass with
putnext.

NOTE: Each stream module has a read and write service (srv) routine. If a ser­
vice routine is not needed (because the put routine processes all messages), a
NULL pointer should be placed in module’s qinit structure. Do not use the
nulldev routine instead of the NULL pointer. Use of nulldev for a srv routine
may result in flow control errors.

SEE ALSO
The BCI Driver Development Guide, Chapter 7, “STREAMS”

bcanput(U3DK), canput(U3DK), getq(U3DK), put(U2DK), putbq(U3DK),
putnext(U3DK), putq(U3DK), queue(U4DK)
NAME
start — start access to a device

SYNOPSIS
void prefixstart();

DESCRIPTION
The start routine is called when a computer starts placing a device into a known state. At the time this routine is called, the developer cannot depend on root being mounted. However, the developer can depend on low level system services being available such as interrupts enabled.

A start routine may perform the following types of activities:
- initialize data structures for device access
- allocate buffers for private buffering scheme
- map device into virtual address space
- initialize hardware (for example, perform a system generation and reset the board)
- initialize the serial device for character drivers
- initialize any static data associated with the driver

SEE ALSO
init(D2DK), cunix(1M), master(4)
strategy (D2DK)  

NAME

strategy – perform block I/O

SYNOPSIS

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

int prefixstrategy(struct buf *bp);

ARGUMENT

bp Pointer to the buf(D4DK) structure.

DESCRIPTION

The strategy routine is called indirectly (through bdevsw) by the kernel to read and write blocks of data on the block device. strategy may also be called directly or indirectly (via a call to the kernel function physiock(D3D)), to support the raw character interface of a block device (read(D2DK), write(D2DK) and ioctl(D2DK)). The strategy routine’s responsibility is to set up and initiate the transfer.

RETURN VALUE

On an error condition, OR the b_flags member of the buf(D4DK) structure with B_ERROR and set the b_error member to the appropriate error value.

SEE ALSO

read(D2DK), write(D2DK)
NAME
write – write data to a device

SYNOPSIS

#include <sys/types.h>
#include <sys/errno.h>
#include <sys/open.h>
#include <sys/cred.h>

int prefixwrite(dev_t dev, uio_t *uio_p, cred_t *cred_p);

ARGUMENTS

dev  Device number.
uio_p Pointer to the uio(D4DK) structure that describes where the data is to be stored in user space.
cred_p Pointer to the cred(D4DK) user credential structure for the I/O transaction.

DESCRIPTION

Used for character or raw data I/O, the driver write routine is called indirectly through cdevsw by the write(2) system call. The write routine supervises the data transfer from user space to a device described by the uio(D4DK) structure.

The write routine should check the validity of the minor number component of dev and the user credentials pointed to by cred_p (if pertinent).

RETURN VALUE

The write routine should return 0 for success, or the appropriate error number. Refer to Appendix A for a list of DDI/DKI error numbers.

SEE ALSO

read(D2DK)
# Kernel Functions (D3)

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3-1</td>
</tr>
<tr>
<td>Function Summary</td>
<td>3-3</td>
</tr>
<tr>
<td>Manual Pages</td>
<td>3-8</td>
</tr>
</tbody>
</table>
Introduction

This chapter describes the kernel functions available for use by device drivers. Each function is described in a separate entry. Most functions are part of both DDI and DKI—these are indicated by the (D3DK) cross reference code. Functions belonging only to DDI are cross-referenced by (D3D) and DKI-only functions are marked (D3K).

The following additional routines were added to D3DK for device driver support on the M68000 family of processors and M88000 family of processors:

- bp_iosetup(D3DK)
- dcache_inval(D3DK)
- dcache_sync(D3DK)
- dma_sgio(D3DK)
- iomapin(D3DK)
- iomem_alloc(D3DK)
- iomem_free(D3DK)
- ioprobe(D3DK)
- mp_iosetup(D3DK)
- uiophysio(D3DK)

In this section, the information for each driver function is organized under the following headings:

- NAME summarizes the function’s purpose.
- SYNOPSIS shows the syntax of the function’s entry point in the source code. #include statements are shown for required header files.
- ARGUMENTS describes any arguments required to invoke the function.
- DESCRIPTION describes general information about the function.
- RETURN VALUE describes the return values and messages that can result from invoking the function.
- LEVEL indicates from which driver level (base or interrupt) the function can be called.
• SEE ALSO indicates functions that are related by usage and sources, and which can be referred to for further information.

• EXAMPLE shows how the function can be used in driver code.

**NOTE**

The `ddi.h` header file undefines macros that have been reimplemented as functions in UNIX System V Release 4. Always place `ddi.h` at the end of the list of `include` statements to avoid contention between macro and function declarations.
### Function Summary

Table 3-1 summarizes the STREAMS functions described in this section. STREAMS functions may be used in either DDI or DKI.

**Table 3-1: STREAMS Kernel Function Summary**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjmsg</td>
<td>remove the specified number of bytes from a message</td>
</tr>
<tr>
<td>allocb</td>
<td>allocate a message block</td>
</tr>
<tr>
<td>backq</td>
<td>get pointer to the previous queue</td>
</tr>
<tr>
<td>bcanput</td>
<td>test for flow control in specified priority band</td>
</tr>
<tr>
<td>bufcall</td>
<td>get buffer when allocb fails</td>
</tr>
<tr>
<td>canput</td>
<td>test for room in a message queue</td>
</tr>
<tr>
<td>copyb</td>
<td>copy a message block</td>
</tr>
<tr>
<td>copymsg</td>
<td>copy a message to a new message</td>
</tr>
<tr>
<td>datamsg</td>
<td>test whether a message is a data message</td>
</tr>
<tr>
<td>dupb</td>
<td>duplicate a message block descriptor</td>
</tr>
<tr>
<td>dupmsg</td>
<td>duplicate a message</td>
</tr>
<tr>
<td>enabledk</td>
<td>enable a queue for service</td>
</tr>
<tr>
<td>esballoc</td>
<td>allocate a message block with a shared buffer</td>
</tr>
<tr>
<td>esbbcall</td>
<td>get message header when esballoc fails</td>
</tr>
<tr>
<td>flushband</td>
<td>flush messages for specified priority band</td>
</tr>
<tr>
<td>flushq</td>
<td>remove messages from a queue</td>
</tr>
<tr>
<td>freeb</td>
<td>free a message block</td>
</tr>
<tr>
<td>freemsg</td>
<td>free all message blocks in a message</td>
</tr>
<tr>
<td>getq</td>
<td>get a message from the front of a queue</td>
</tr>
<tr>
<td>insq</td>
<td>insert a message into a queue</td>
</tr>
<tr>
<td>linkb</td>
<td>concatenate two message blocks</td>
</tr>
<tr>
<td>mp_iosetup</td>
<td>create scatter/gather list</td>
</tr>
</tbody>
</table>
### Table 3-1: STREAMS Kernel Function Summary (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgdsize</td>
<td>return the number of bytes in a message</td>
</tr>
<tr>
<td>noenable</td>
<td>prevent a queue from being scheduled</td>
</tr>
<tr>
<td>OTHERQ</td>
<td>get a pointer to a module’s other queue</td>
</tr>
<tr>
<td>pullupmsg</td>
<td>concatenate bytes in a message</td>
</tr>
<tr>
<td>putbq</td>
<td>place a message at the head of a queue</td>
</tr>
<tr>
<td>putctl</td>
<td>put a control message on a queue</td>
</tr>
<tr>
<td>putctll</td>
<td>put a control message with a one-byte parameter on a queue</td>
</tr>
<tr>
<td>putnext</td>
<td>send a message to the next module in the stream</td>
</tr>
<tr>
<td>putq</td>
<td>put a message on a queue</td>
</tr>
<tr>
<td>qenable</td>
<td>enable a queue</td>
</tr>
<tr>
<td>qreply</td>
<td>send a message in the reverse direction</td>
</tr>
<tr>
<td>qsize</td>
<td>find the number of messages on a queue</td>
</tr>
<tr>
<td>RD</td>
<td>get a pointer to a module’s read queue</td>
</tr>
<tr>
<td>rmvb</td>
<td>remove a message block from a queue</td>
</tr>
<tr>
<td>rmvq</td>
<td>remove a message from a queue</td>
</tr>
<tr>
<td>SAMESTR</td>
<td>test if next queue is same type</td>
</tr>
<tr>
<td>strlog</td>
<td>submit messages for logging</td>
</tr>
<tr>
<td>strqget</td>
<td>get information about a queue</td>
</tr>
<tr>
<td>strqset</td>
<td>change information about a queue</td>
</tr>
<tr>
<td>testb</td>
<td>check for an available buffer</td>
</tr>
<tr>
<td>unlinkb</td>
<td>remove the message block from the head of a message</td>
</tr>
<tr>
<td>WR</td>
<td>get pointer to this module’s write queue</td>
</tr>
</tbody>
</table>
Table 3-2 summarizes the functions not specific to STREAMS. Functions can be used in either DDI or DKI, except as noted.

**Table 3-2: Kernel Functions Not Specific to STREAMS**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcopy</td>
<td>copy data between locations in the kernel, for example, from one buffer to another</td>
<td></td>
</tr>
<tr>
<td>biodone</td>
<td>release buffer after block I/O and wakeup processes</td>
<td></td>
</tr>
<tr>
<td>biowait</td>
<td>suspend processes pending completion of block I/O</td>
<td></td>
</tr>
<tr>
<td>bp_iosetup</td>
<td>create scatter/gather list</td>
<td></td>
</tr>
<tr>
<td>bp_mapin</td>
<td>allocate virtual address space</td>
<td></td>
</tr>
<tr>
<td>bp_mapout</td>
<td>deallocate virtual address space</td>
<td></td>
</tr>
<tr>
<td>brelse</td>
<td>return buffer to the kernel</td>
<td></td>
</tr>
<tr>
<td>btop</td>
<td>return number of memory pages contained in specified number of bytes (downward rounding)</td>
<td></td>
</tr>
<tr>
<td>btopr</td>
<td>return number of memory pages contained in specified number of bytes (upward rounding)</td>
<td></td>
</tr>
<tr>
<td>bzero</td>
<td>clear memory for a number of bytes</td>
<td></td>
</tr>
<tr>
<td>clrbuf</td>
<td>erase buffer contents</td>
<td></td>
</tr>
<tr>
<td>cmn_err</td>
<td>display message or panic the system</td>
<td></td>
</tr>
<tr>
<td>copyin</td>
<td>copy data from user space to the driver</td>
<td></td>
</tr>
<tr>
<td>copyout</td>
<td>copy data from the driver to user space</td>
<td></td>
</tr>
<tr>
<td>dcache_inval</td>
<td>invalidate data cache</td>
<td></td>
</tr>
<tr>
<td>dcache_sync</td>
<td>sync the data cache</td>
<td></td>
</tr>
<tr>
<td>delay</td>
<td>delay for specified number of clock ticks</td>
<td></td>
</tr>
<tr>
<td>dma_pageio</td>
<td>break up DMA requests</td>
<td>DDI only</td>
</tr>
<tr>
<td>dma_sgio</td>
<td>break up scatter/gather request</td>
<td></td>
</tr>
<tr>
<td>drv_getparm</td>
<td>retrieve kernel state information</td>
<td></td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>drv_hztousec</td>
<td>convert from clock ticks to microseconds</td>
<td></td>
</tr>
<tr>
<td>drv_priv</td>
<td>determine driver privileges</td>
<td></td>
</tr>
<tr>
<td>drv_usecotohz</td>
<td>convert from microseconds to clock ticks</td>
<td></td>
</tr>
<tr>
<td>drv_usecwait</td>
<td>wait for specified number of microseconds</td>
<td></td>
</tr>
<tr>
<td>etoimajor</td>
<td>convert external major number to internal major number</td>
<td></td>
</tr>
<tr>
<td>freerbuf</td>
<td>free a raw buffer header</td>
<td></td>
</tr>
<tr>
<td>getemajor</td>
<td>get external major number</td>
<td>DDI only</td>
</tr>
<tr>
<td>geteminor</td>
<td>get external minor number</td>
<td>DDI only</td>
</tr>
<tr>
<td>geterror</td>
<td>return an I/O error</td>
<td></td>
</tr>
<tr>
<td>getmajor</td>
<td>get major number</td>
<td></td>
</tr>
<tr>
<td>getminor</td>
<td>get minor number</td>
<td></td>
</tr>
<tr>
<td>getrbuf</td>
<td>get a raw buffer header</td>
<td></td>
</tr>
<tr>
<td>getvec</td>
<td>get an interrupt vector for a given virtual board address</td>
<td>DDI only</td>
</tr>
<tr>
<td>hat_getkpfnum</td>
<td>get page frame number for address</td>
<td>DKI only</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>initialize error logging in the hard disk</td>
<td>DDI only</td>
</tr>
<tr>
<td>hdelog</td>
<td>log a hard disk error</td>
<td>DDI only</td>
</tr>
<tr>
<td>iomapin</td>
<td>map an I/O address in</td>
<td></td>
</tr>
<tr>
<td>iomem_alloc</td>
<td>allocate physically contiguous memory</td>
<td></td>
</tr>
<tr>
<td>iomem_free</td>
<td>free memory allocated by iomem_alloc</td>
<td></td>
</tr>
<tr>
<td>ioprobe</td>
<td>probe I/O address</td>
<td></td>
</tr>
<tr>
<td>itoemajor</td>
<td>internal major number to external number</td>
<td>DDI only</td>
</tr>
<tr>
<td>kmem_alloc</td>
<td>allocate from kernel free space</td>
<td></td>
</tr>
<tr>
<td>kmem_free</td>
<td>free previously allocated kernel memory</td>
<td></td>
</tr>
<tr>
<td>kmem_zalloc</td>
<td>allocate and clear storage from kernel free memory</td>
<td></td>
</tr>
<tr>
<td>kvtophys</td>
<td>convert kernel virtual to physical address</td>
<td>DDI only</td>
</tr>
<tr>
<td>makedevice</td>
<td>create a device number</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-2: Kernel Functions Not Specific to STREAMS (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>return the larger of two integers</td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>return the smaller of two integers</td>
<td></td>
</tr>
<tr>
<td>page_numtopp</td>
<td>convert page frame number to page structure</td>
<td></td>
</tr>
<tr>
<td>page_pptonum</td>
<td>convert page structure to page frame number</td>
<td></td>
</tr>
<tr>
<td>physiock</td>
<td>validate and issue raw I/O request</td>
<td>DDI only</td>
</tr>
<tr>
<td>pollwakeup</td>
<td>inform a process that an event has occurred</td>
<td></td>
</tr>
<tr>
<td>ptob</td>
<td>convert size in pages to size in bytes</td>
<td></td>
</tr>
<tr>
<td>rmalloc</td>
<td>allocate space from a private space management map</td>
<td></td>
</tr>
<tr>
<td>rmfree</td>
<td>free space back into a private space management map</td>
<td></td>
</tr>
<tr>
<td>rminit</td>
<td>initialize a private space management map</td>
<td></td>
</tr>
<tr>
<td>rmsetwant</td>
<td>set the map's wait flag for wakeup</td>
<td></td>
</tr>
<tr>
<td>rmwant</td>
<td>wait for free memory</td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution</td>
<td></td>
</tr>
<tr>
<td>spl</td>
<td>suspend or allow interrupts</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>call function in clock ticks</td>
<td></td>
</tr>
<tr>
<td>uiomove</td>
<td>copy kernel data using uio structure</td>
<td></td>
</tr>
<tr>
<td>uiophysio</td>
<td>validate and issue raw I/O request</td>
<td></td>
</tr>
<tr>
<td>untimout</td>
<td>cancel timeout with matching ID</td>
<td></td>
</tr>
<tr>
<td>ureadc</td>
<td>add character to uio structure</td>
<td></td>
</tr>
<tr>
<td>useracc</td>
<td>verify user access to data structures</td>
<td></td>
</tr>
<tr>
<td>uwritec</td>
<td>remove a character from a uio structure</td>
<td></td>
</tr>
<tr>
<td>vtop</td>
<td>convert virtual to physical address</td>
<td>DDI only</td>
</tr>
<tr>
<td>wakeup</td>
<td>resume suspended execution</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

## 3. Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjmsg(D3DK)</td>
<td>trim bytes from a message</td>
</tr>
<tr>
<td>allocb(D3DK)</td>
<td>allocate a message block</td>
</tr>
<tr>
<td>backq(D3DK)</td>
<td>get pointer to the queue behind the current queue</td>
</tr>
<tr>
<td>bcanput(D3DK)</td>
<td>test for flow control in specified priority band</td>
</tr>
<tr>
<td>bcopy(D3DK)</td>
<td>copy data between address locations in the kernel</td>
</tr>
<tr>
<td>biodone(D3DK)</td>
<td>release buffer after block I/O and wakeup processes</td>
</tr>
<tr>
<td>biowait(D3DK)</td>
<td>suspend processes pending completion of block I/O</td>
</tr>
<tr>
<td>bp_iostatup(D3DK)</td>
<td>create scatter/gather list for block drivers</td>
</tr>
<tr>
<td>bp_mapin(D3DK)</td>
<td>allocate virtual address space</td>
</tr>
<tr>
<td>bp_mapout(D3DK)</td>
<td>deallocate virtual address space</td>
</tr>
<tr>
<td>brelse(D3DK)</td>
<td>return buffer to the bfreelist</td>
</tr>
<tr>
<td>btop(D3DK)</td>
<td>convert size in bytes to size in pages (round down)</td>
</tr>
<tr>
<td>btopr(D3DK)</td>
<td>convert size in bytes to size in pages (round up)</td>
</tr>
<tr>
<td>bufcall(D3DK)</td>
<td>call a function when a buffer becomes available</td>
</tr>
<tr>
<td>bzero(D3DK)</td>
<td>clear memory for a given number of bytes</td>
</tr>
<tr>
<td>canput(D3DK)</td>
<td>test for room in a message queue</td>
</tr>
<tr>
<td>clrbuf(D3DK)</td>
<td>erase the contents of a buffer</td>
</tr>
<tr>
<td>cmn_err(D3DK)</td>
<td>display an error message or panic the system</td>
</tr>
<tr>
<td>copyb(D3DK)</td>
<td>copy a message block</td>
</tr>
<tr>
<td>copyin(D3DK)</td>
<td>copy data from a user program to a driver buffer</td>
</tr>
<tr>
<td>copymsg(D3DK)</td>
<td>copy a message</td>
</tr>
<tr>
<td>copyout(D3DK)</td>
<td>copy data from a driver to a user program</td>
</tr>
<tr>
<td>datamsg(D3DK)</td>
<td>test whether a message is a data message</td>
</tr>
<tr>
<td>delay(D3DK)</td>
<td>delay process execution for a specified number of clock ticks</td>
</tr>
<tr>
<td>dcache_inval(D3DK)</td>
<td>invalidate the data cache</td>
</tr>
<tr>
<td>dcache_sync(D3DK)</td>
<td>sync the data cache</td>
</tr>
<tr>
<td>dma_pageio(D3D)</td>
<td>break up an I/O request into manageable units</td>
</tr>
<tr>
<td>dma_sgio(D3D)</td>
<td>break up an I/O request for controller that does scatter/gather</td>
</tr>
<tr>
<td>drv_getparm(D3DK)</td>
<td>retrieve kernel state information</td>
</tr>
<tr>
<td>drv_hztousec(D3DK)</td>
<td>convert clock ticks to microseconds</td>
</tr>
<tr>
<td>drv_priv(D3DK)</td>
<td>determine driver privilege</td>
</tr>
<tr>
<td>drv_usectohz(D3DK)</td>
<td>convert microseconds to clock ticks</td>
</tr>
<tr>
<td>drv_usecwait(D3DK)</td>
<td>busy-wait for specified interval</td>
</tr>
</tbody>
</table>
2. System Calls

dump(D2DK) .............................................................. gain access to a device crash dump routine
dupb(D3DK) ...................................................................... duplicate a message block descriptor
dupmsg(D3DK) .................................................................. duplicate a message
enableok(D3DK) .......................................................... reschedule a queue for service
esballoc(D3DK) ................................................................ allocate a message block using a shared buffer
esbbcall(D3DK) ................................................................ call function when buffer is available
etomajor(D3D) .......................................................... convert external to internal major device number
flushband(D3DK) ...................................................... flush messages for a specified priority band
flushq(D3DK) .................................................................................. remove messages from a queue
freeb(D3DK) .......................................................................................... free a message block
freemsg(D3DK) ....................................................................... free all message blocks in a message
freerbuf(D3DK) ................................................................................................ free a raw buffer header
gatemajor(D3D) .......................................................... get external major device number
getminor(D3D) ........................................................ get external minor device number
geterror(D3DK) .......................................................................................................... return 1/0 error
getmajor(D3DK) .......................................................... get major or internal major device number
getminor(D3DK) ........................................................ get minor or internal minor device number
getq(D3DK) ............................................................................... get the next message from a queue
getbuf(D3DK) .............................................................................................. get a raw buffer header
hat_getakpfnum(D3K) .................................................. get page frame number for kernel address
hdeeqd(D3D) ................................................................................. initialize hard disk error logging
hdelog(D3D) .......................................................................................................... log hard disk error
iomapin(D3DK) ................................................................................... map an I/O address (device)
imem_alloc(D3DK) ................................................................ allocate physically contiguous memory
imem_free(D3DK) .......................................................................................... free memory allocated by imem_alloc
ioprobe(D3DK) ........................................................................... probe an I/O address for a device
insq(D3DK) ......................................................................................... insert a message into a queue
itoemajor(D3D) ................................................ convert internal to external major device number
kmem_alloc(D3DK) .......................................................... allocate space from kernel free memory
kmem_free(D3DK) .......................................................................................... free previously allocated kernel memory
kmem_zalloc(D3DK) .................................................. allocate and clear space from kernel free memory
kvtophys(D3D) .......................................................... convert kernel virtual address to physical address
linkb(D3DK) ........................................................................... concatenate two message blocks
makedevice(D3DK) .......................................................... make device number from external major and minor
max(D3DK) .......................................................................................................................... return the larger of two integers
mp_iosetup(D3DK) ............................................. create scatter/gather list for STREAMS drivers
min(D3DK) .......................................................................................................................... return the lesser of two integers
msgdsize(D3DK) ............................................................. return the number of bytes in a message
noenable(D3DK) ................................................................. prevent a queue from being scheduled
OTHERQ(D3DK) ................................................................. get pointer to queue's partner queue
page_numtopp(D3DK) ................................................................. convert page frame number to page structure
page_pptonum(D3DK) ................................................................. convert page structure to page frame number
physiock(D3DK) ................................................................. validate and issue raw I/O request
pollwakeup(D3DK) ................................................................. inform a process that an event has occurred
ptob(D3DK) ................................................................. convert size in pages to size in bytes
pullupmsg(D3DK) ................................................................. concatenate bytes in a message
putbq(D3DK) ................................................................. place a message at the head of a queue
putc1(D3DK) ................................................................. send a control message to a queue
putct1(D3DK) ................................................................. send a control message with a one-byte parameter to a queue
putnext(D3DK) ................................................................. send a message to the next queue
putq(D3DK) ................................................................. put a message on a queue
qenable(D3DK) ................................................................. enable a queue
qreply(D3DK) ................................................................. send a message on a stream in the reverse direction
qsize(D3DK) ................................................................. find the number of messages on a queue
RD(D3DK) ................................................................. get pointer to the read queue
rmalloc(D3DK) ................................................................. allocate space from a private space management map
rmfree(D3DK) ................................................................. release free space back into a private space management map
rminit(D3DK) ................................................................. initialize a private space management map
rmsetwant(D3DK) ................................................................. set the map's wait flag for a wakeup
rmvb(D3DK) ................................................................. remove a message block from a message
rmvq(D3DK) ................................................................. remove a message from a queue
rmwant(D3DK) ................................................................. wait for free memory
SAMESTR(D3DK) ................................................................. test if next queue is same type
sleep(D3DK) ................................................................. suspend process activity pending execution of an event
spl(D3DK) ................................................................. block/allow interrupts
strlog(D3DK) ................................................................. submit messages to the log driver
strqget(D3DK) ................................................................. get information about a queue or band of the queue
strqset(D3DK) ................................................................. change information about a queue or band of the queue
testb(D3DK) ................................................................. check for an available buffer
timeout(D3DK) ................................................................. execute a function after a specified length of time
uiomove(D3DK) ................................................................. copy kernel data using uio(D4DK) structure
uio physio(D3DK) ................................................................. validate and issue raw I/O request
unlinkb(D3DK) ................................................................. remove a message block from the head of a message
unt imeout(D3DK) ................................................................. cancel previous timeout(D3DK) function call
ureadc(D3DK) ................................................................. add character to a uio structure
useracc(D3DK) ................................................................. verify whether user has access to memory
uwritec(D3DK) ................................................................. remove a character from a uio structure
vtop(D3DK) ................................................................. convert virtual to physical address
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wakeup(D3DK)</td>
<td>resume suspended process execution</td>
</tr>
<tr>
<td>WR(D3DK)</td>
<td>get pointer to the write queue for this module or driver</td>
</tr>
</tbody>
</table>
NAME
adjmsg - trim bytes from a message

SYNOPSIS
#include <sys/stream.h>

int adjmsg(mblk_t *mp, int len);

ARGUMENTS
*mp Pointer to the message to be trimmed.
len The number of bytes to be removed.

DESCRIPTION
adjmsg removes bytes from a message. |len| (the absolute value of len) specifies how many bytes are to be removed. If len is greater than 0, bytes are removed from the head of the message. If len is less than 0, bytes are removed from the tail. adjmsg fails if |len| is greater than the number of bytes in mp.

RETURN VALUE
If the message can be trimmed successfully, 1 is returned. Otherwise, 0 is returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb (DDI/DKI(STREAMS))

NAME
allocb — allocate a message block

SYNOPSIS
#include <sys/stream.h>

mblk_t *allocb(int size, int pri);

ARGUMENTS
size The number of bytes in the message block.
pri Priority of the request (no longer used).

DESCRIPTION
allocb tries to allocate a STREAMS message block. Buffer allocation fails only when the system is out of memory. If no buffer is available, the bufcall(D3DK) function can help a module recover from an allocation failure.

NOTE: The pri argument is no longer used in UNIX System V Release 4, but is retained for compatibility with existing drivers.

The following figure identifies the data structure members that are affected when a message block is allocated.

```
message block (mblk_t)  data block (dblkt_t)  data buffer
b_cont (0)               db_base
b_rptr db_lim
b_wptr db_type (M_DATA)
bd_start (5) db_class (f)
```

RETURN VALUE
If successful, allocb returns a pointer to the allocated message block of type M_DATA (defined in sys/stream.h). If a block cannot be allocated, a NULL pointer is returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
bufcall(D3DK), esballoc(D3DK), esbbcall(D3DK), testb(D3DK)

EXAMPLE
Given a pointer to a queue (q) and an error number (err), the send_error routine sends an M_ERROR type message to the stream head.

If a message cannot be allocated, 0 is returned, indicating an allocation failure (line 8). Otherwise, the message type is set to M_ERROR (line 10). Line 11 increments the write pointer (bp->b_wptr) by the size (one byte) of the data in the message.
A message must be sent up the read side of the stream to arrive at the stream head. To determine whether \( q \) points to a read queue or a write queue, the \( q->q\_flag \) member is tested to see if QREADR is set (line 13). If it is not set, \( q \) points to a write queue, and in line 14 the RD(D3DK) function is used to find the corresponding read queue. In line 15, the putnext(D3DK) function is used to send the message upstream, returning 1 if successful.

```c
1   send_error(q,err)
2       queue_t *q;
3       unsigned char err;
4 {
5       mblk_t *bp;
6
7       if ((bp = allocb(l, BPRI_HI)) == NULL) /* allocate msg. block */
8           return(0);
9
10      bp->b_datap->db_type = M_ERROR; /* set msg type to M_ERROR */
11      *bp->b_wptr++ = err; /* increment write pointer */
12
13      if(!q->q_flag & QREADR)) /* if not read queue */
14          q = RD(q); /* get read queue */
15      putnext(q,bp); /* send message upstream */
16      return(1);
17 }
```
NAME
backq - get pointer to the queue behind the current queue

SYNOPSIS
#include <sys/stream.h>
queue_t *backq(queue_t *cq);

ARGUMENT
*cq The pointer to the current queue. queue_t is an alias for the
queue(D4DK) structure.

DESCRIPTION
backq returns a pointer to the queue preceding cq (the current queue). If cq is a
read queue, backq returns a pointer to the queue downstream from cq, unless it
is the stream end. If cq is a write queue, backq returns a pointer to the next
queue upstream from cq, unless it is the stream head.

RETURN VALUE
If successful, backq returns a pointer to the queue preceding the current queue.
Otherwise, it returns NULL.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
NAME
bcanput - test for flow control in specified priority band

SYNOPSIS
#include <sys/stream.h>
int bcanput(queue_t *q, unsigned char pri);

ARGUMENT
q Pointer to the message queue.
pri Message priority.

DESCRIPTION
Like the canput(D3DK) function, bcanput searches through the stream (starting
at q) until it finds a queue containing a service routine where the message can be
enqueued, or until it reaches the end of the stream. If found, the queue contain-
ing the service routine is tested to see if there is room for a message in the queue.
If the queue is full, bcanput sets the QWANTW flag to back-enable the caller’s
service routine.

If pri is 0, the bcanput call is equivalent to a call to canput.

NOTE: You are responsible for both testing a queue with bcanput and refraining
from placing a message on the queue if bcanput fails.

RETURN VALUE
A 1 is returned if a message of priority pri can be placed on the queue, or if the
band does not yet exist on the queue. A 0 is returned if the priority band is
flow-controlled.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
canput(D3DK), putbq(D3DK), putnext(D3DK)
NAME
bcopy — copy data between address locations in the kernel

SYNOPSIS
#include <sys/types.h>

int bcopy(caddr_t from, caddr_t to, long bcount);

ARGUMENTS
from Source address from which the copy is made.
to Destination address to which copy is made.
bcount The number of bytes moved.

DESCRIPTION
bcopy copies bcount bytes from one kernel address to another. If the input and
output addresses overlap, the command executes, but the results may not be as
expected.

CAUTION: The from and to addresses must be within the kernel space. No range
checking is done. If an address outside of the kernel space is selected, the driver
may corrupt the system in an unpredictable way.

Note that bcopy should never be used to move data in or out of a user buffer,
because it has no provision for handling page faults. The user address space
can be swapped out at any time, and bcopy always assumes that there will be no
paging faults. If bcopy attempts to access the user buffer when it is swapped
out, the system will panic. It is safe to use bcopy to move data within kernel
space, since kernel space is never swapped out.

RETURN VALUE
Under all conditions, 0 is returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 6, "Input/Output Operations"
copyin(D3DK), copyout(D3DK)

EXAMPLE
An I/O request is made for data stored in a RAM disk. If the I/O operation is a
read request, the data is copied from the RAM disk to a buffer (line 7). If it is a
write request, the data is copied from a buffer to the RAM disk (line 11). The
bcopy function is used since both the RAM disk and the buffer are part of the
kernel address space.

```c
#define RAMDNBLK 1000 /* blocks in the RAM disk */
#define RAMDBSIZ 512 /* bytes per block */

char ramdbks[RAMDNBLK][RAMDBSIZ]; /* blocks forming RAM disk */

... 

if (bp->b_flags & B_READ) /* if read request, copy data from RAM */
    /* disk data block to system buffer */
    bcopy(&ramdbks[bp->b_blkno][0], bp->b_un.b_addr, bp->b_bcount);
```

3/91
9 else /* else write request, copy data from a */
10 /* system buffer to RAM disk data block */
11 bcopy(bp->b_un.b_addr, &ramdblk[bp->b_bkno][0], bp->b_bcount);
NAME
biodone - release buffer after block I/O and wakeup processes

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

void biodone(struct buf *bp);

ARGUMENT
*bp  Pointer to the buffer header structure defined in buf.h. This is the address of the buffer header associated with the buffer where the I/O occurred.

DESCRIPTION
The biodone function is called by either the driver int(D2D) or strategy(D2DK) routines when a block I/O request is complete. In general, biodone awakens sleeping processes waiting for the I/O to complete, sets the B_DONE flag in the buf structure b_flags field, and releases the block if the I/O is asynchronous.

For drivers that wish to make multiple I/O requests without releasing and reallocating a buffer header for each individual request, biodone provides the capability to check for an additional function to be called before the buffer header is released. Additional routines to be called from biodone are referenced by the (*b_biodone) field of the buf structure.

biodone performs the following functions in the order presented:
- checks the (*biodone) field of the buf structure for additional routines to be called. If an additional routine is referenced, it is called and the functions listed below are not completed.
- awakens the process(es) that called sleep(D3DK) to wait for the buffer header if I/O is synchronous
- releases the block if I/O is asynchronous and awakens processes awaiting asynchronous I/O
- marks b_flags of buffer with B_DONE

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 9, "Synchronizing Hardware and Software Events"
biowait(D3DK), buf(D4DK), delay(D3DK), int(D3D), strategy(D3DK), sleep(D3DK), timeout(D3DK), untimeout(D3DK), wakeup(D3DK)

EXAMPLE
Generally, the first validation test performed by any block device strategy(D2DK) routine is a check for an end-of-file (EOF) condition. The strategy routine is responsible for determining an EOF condition when the device is accessed directly. If a read request is made for one block beyond the
limits of the device (line 10), it will report an EOF condition. Otherwise, if the
request is outside the limits of the device, the routine will report an error condi-
tion. In either case, report the I/O operation as complete (line 27).

```c
#define RAMDSNBLK 1000  /* Number of blocks in RAM disk */
#define RAMDSNBSIZ 512   /* Number of bytes per block */
char ramdblks[RAMDSNBLK][RAMDSNBSIZ]; /* Array containing RAM disk */

ramdstrategy(bp)
register struct buf *bp;
{
    register daddr_t blkno = bp->b_blkno; /* get block number */
    if (blkno < 0 || blkno >= RAMDSNBLK) {
        /*
         * If requested block is outside RAM disk
         * limits, test for EOF which could result
         * from a direct (physlock) request.
         */
        if (blkno == RAMDSNBLK && bp->b_flags & B_READ) {
            /*
             * If read is for block beyond RAM disk
             * limits, mark EOF condition.
             */
            bp->b_resid -= bp->b_bcount; /* compute return value */
        } else { /* I/O attempt is beyond */
            bp->b_error = ENXIO; /* limits of RAM disk */
            bp->b_flags |= B_ERROR; /* return error */
        } /* endif */
    } /* endif */
    biodone(bp); /* mark I/O complete (B_DONE) */
    /*
     * Wake any processes awaiting this I/O
     * or release buffer for asynchronous
     * (B_ASYNC) request.
     */
    return;
} /* endif */
```
NAME
biowait - suspend processes pending completion of block I/O

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

int biowait(struct buf *bp);

ARGUMENT
*bp Pointer to the buf structure.

DESCRIPTION
The biowait function suspends process execution during a block I/O transfer by
calling sleep(D3DK). Block driver routines using the buf structure to allocate
buffers can use the biowait function to suspend a process while waiting for a
read or write request to complete.

The biowait function is one of three functions used to aid block I/O transfers.
The other functions in this group are biodone(D3DK), which notifies biowait
that the I/O is complete, and brelse, which frees the buffer allocated for the
transfer.

Drivers using the biowait function must also include the biodone(D3DK) func-
tion in their interrupt routines. The biodone function awakens biowait when
the I/O transfer is complete.

Because biowait calls sleep, biowait cannot be called from an interrupt routine
or from an init(D2D) routine.

RETURN VALUE
None. However, biowait returns any error that may have occurred during the
I/O transfer to the user using geterror(D3DK).

LEVEL
Base Only (Do not call from an interrupt routine)

SEE ALSO
biodone(D3DK), brelse(D3DK), sleep(D3DK), timeout(D3DK),
unttimeout(D3DK), wakeup(D3DK)
NAME
bp_iosetup - create scatter/gather list for block drivers

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>

int bp_iiosetup(struct buf *bp, struct iolist *list, int size, int maxcoalesce);

ARGUMENTS
*bp pointer to the buffer header structure
*list pointer to the scatter/gather list
size the number of I/O vectors in list
maxcoalesce the maximum coalescing size the controller can handle

DESCRIPTION
bp_iiosetup is invoked from block driver strategy routines. It returns a count of at most size (physical address, length) I/O vectors which are returned in list, and can then be passed to a controller. The I/O vectors are computed from the virtual address or page list and count in the buffer header. This routine ensures that the data cache has been invalidated or synced for all pages involved in the DMA transfer. bp_iiosetup fails with a return value of -1 if list becomes larger than size.

This routine is provided specifically for device driver support on the M68000 or M88000 family of processors.

RETURN VALUE
The number of entries in list.

LEVEL
Base.

SEE ALSO
mp_iiosetup(D3DK), buf(D4DK)
NAME
bp_mapin - allocate virtual address space

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
vaddr_t bp_mapin(struct buf *bp);

ARGUMENTS
bp pointer to the buffer header structure

DESCRIPTION
bp_mapin is used to map virtual address space to a page list maintained by the
buffer header during a paged-I/O request. bp_mapin allocates system virtual
address space, maps that space to the page list, and returns the offset into the
map. The offset is stored in the bp->b_un.b_addr field of the buf structure (see
buf(D4DK)). Virtual address space is then deallocated using the bp_mapout func­tion.

If a NULL page list is encountered, bp_mapin returns without allocating space and
no mapping is performed.

bp_mapin should be used by drivers that map the address specified in the buf
pointer into kernel space (that is, drivers that access data in the buffer rather than
just handing an address to the controller). This routine ensures that the data
cache has been invalidated or sync'ed for all page(s) involved in the DMA
transfer.

RETURN VALUE
The starting address of the allocated system virtual address space.

LEVEL
Base.

SEE ALSO
bp_mapout(D3DK), buf(D4DK)
NAME
    bp_mapout - deallocate virtual address space

SYNOPSIS
    #include <sys/types.h>
    #include <sys/buf.h>
    void bp_mapin(struct buf *bp);

ARGUMENTS
    bp                       Pointer to the buffer header structure.

DESCRIPTION
    This function deallocates system virtual address space allocated by a previous call
to bp_mapin(D3DK). bp_mapin maps virtual address space to a page list main­
tained by the buffer header for a paged-I/O request, then returns the offset into
the map to the b_addr field of the buf structure.

This routine ensures that the data cache has been invalidated or sync'ed for all
page(s) involved in the DMA transfer.

RETURN VALUE
    None

LEVEL
    Base

SEE ALSO
    bp_mapin(D3DK), buf(D4DK)
NAME
brelse - return buffer to the bfreelist

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

void brelse(struct buf *bp);

ARGUMENT
*bp Pointer to the buf structure.

DESCRIPTION
The brelse function returns a previously allocated buffer to the buffer free list. First, brelse wakes up processes sleeping on the buffer. After the driver function is finished with the buffer, brelse returns the buffer header to a list of free buffers and awakens any processes that called sleep(D3DK) to wait for a free buffer on the bfreelist.

RETURN VALUE
None, however, if b_flags has B_ERROR enabled due to an error in an earlier I/O transfer, b_flags is ORed with B_STALE and B_AGE, B_ERROR and B_DELWRI are disabled, and b_error is set to 0.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 6, "Input/Output Operations"
clrbuf(D3DK), biodone(D3DK), biowait(D3DK)
NAME
btop – convert size in bytes to size in pages (round down)

SYNOPSIS
#include <sys/ddi.h>
unsigned long btop(unsigned long nbytes);

ARGUMENT
numbytes Number of bytes.

DESCRIPTION
The btop function returns the number of memory pages that are contained in the
specified number of bytes, with downward rounding in the case that the byte
count is not a page multiple. For example, if the page size is 2048, then
btop(4096) returns 2, and btop(4097) returns 2 as well. btop(0) returns 0.

RETURN VALUE
The return value is always the number of pages. There are no invalid input
values, and therefore no error return values.

LEVEL
Base or Interrupt

SEE ALSO
btopr(D3DK), ptob(D3DK)
NAME
btopr - convert size in bytes to size in pages (round up)

SYNOPSIS
#include <sys/ddi.h>
unsigned long btopr(unsigned long numbytes);

ARGUMENT
numbytes Number of bytes.

DESCRIPTION
This function returns the number of memory pages contained in the specified
number of bytes memory, rounded up to the next whole page. For example, if
the page size is 2048, then btopr(4096) returns 2, and btopr(4097) returns 3.

RETURN VALUE
The return value is always the number of pages. There are no invalid input
values, and therefore no error return values.

LEVEL
Base or Interrupt

SEE ALSO
btop(D3DK), ptob(D3DK)
NAME
bufcall - call a function when a buffer becomes available

SYNOPSIS
#include <sys/stream.h>
int bufcall(int size, int pri, int (*func)(), long arg);

ARGUMENTS
size Number of bytes in the buffer.
pri Priority of the allocb(D3DK) allocation request (not used).
func Function or driver routine to be called when a buffer becomes available.
arg Argument to the function to be called when a buffer becomes available.

DESCRIPTION
bufcall serves as a timeout(D3DK) call of indeterminate length. When a buffer allocation request fails, bufcall can be used to schedule the routine func, to be called with the argument arg when a buffer becomes available. func may be a routine that calls bufcall or it may be another kernel function.

NOTE: Even when func is called by bufcall, allocb(D3DK) can still fail if another module or driver had allocated the memory before func was able to call allocb.

RETURN VALUE
If the bufcall scheduling fails, func is never called and 0 is returned. If successful, bufcall returns 1.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), esbaloc(D3DK), esbbcall(D3DK), testb(D3DK), timeout(D3DK)

EXAMPLE
The purpose of this srv(D2DK) service routine is to add a header to all M_DATA messages. Service routines must process all messages on their queues before returning, or arrange to be rescheduled.

While there are message to be processed (line 13), check to see if it is a high priority message or a normal priority message that can be sent on (line 14). Normal priority message that cannot be sent are put back on the message queue (line 34). If the message was a high priority one, or if was normal priority and canput(D3DK) succeeded, then send all but M_DATA messages to the next stream entity with putnext(D3DK) (line 16).

For M_DATA messages, try to allocate a buffer large enough to hold the header (line 18). If no such buffer is available, the service routine must be rescheduled for a time when a buffer is available. The original message is put back on the queue (line 20) and bufcall (line 21) is used to attempt the rescheduling. It will
bufcall (D3DK)  

DDI/DKI (STREAMS)  

bufcall (D3DK)

succeed if a buffer of the specified size (sizeof (struct hdr)) is available. If it does, qenable(D3DK) will put q on the list of queues to have their service routines called. If bufcall fails, timeout(D3DK) (line 22) is used to try again in about a half second (HZ/2).

If the buffer allocation was successful, initialize the header (lines 25-28), make the message type M_PROTO (line 29), link the M_DATA message to it (line 30), and pass it on (line 31).

```c
1 struct hdr {
2     unsigned int h_size;
3     int h_version;
4 };
5
6 modsrv(q)
7     queue_t *q;
8 |
9     mblk_t *bp;
10     mblk_t *mp;
11     struct hdr *hp;
12
13 while ((mp = getq(q)) != NULL) {
14     if (mp->b_datap->db_type > QPCTL) /* if high priority */
15         canput(q->q_next)) {
16             if (mp->b_datap->db_type == M_DATA)
17                 putnext(q, mp);     /* send all but M_DATA */
18             else {
19                 bp = allocb(sizeof(struct hdr), BPRI_LO);
20                 if (bp == NULL) {
21                     putbq(q, mp);     /* put it back */
22                     if (!bufcall(sizeof(struct hdr), BPRI_LO,
23                         qenable, (long) q) J /* try to reschedule */
24                         timeout(qenable, (long) q, HZ/2);
25                     return;
26                 }
27                 hp = (struct hdr *)bp->b_wptr;
28                 hp->h_size = msgdsz(mp); /* initialize header */
29                 hp->h_version = 1;
30                 bp->b_wptr += sizeof(struct hdr);
31                 bp->b_datap->db_type = M_PROTO; /* make M_PROTO */
32                 bp->b_cont = mp;     /* link it */
33                 putnext(q, bp);     /* pass it on */
34             } else {
35                 putbq(q, mp);     /* put back on the message queue */
36                 return;
37             }
38         }
```
NAME
bzero - clear memory for a given number of bytes

SYNOPSIS
#include <sys/types.h>
int bzero(caddr_t addr, int bytes);

ARGUMENTS
addr Starting virtual address of memory to be cleared.
bytes The number of bytes to clear starting at addr.

DESCRIPTION
The bzero function clears a contiguous portion of memory by filling the memory
with zeros.

CAUTION: The address range specified must be within the kernel space. No
range checking is done. If an address outside of the kernel space is selected, the
driver may corrupt the system in an unpredictable way.

RETURN VALUE
Under normal conditions, a 0 is returned. Otherwise, a -1 is returned.

LEVEL
Base or Interrupt

SEE ALSO
bcopy(D3DK), clrbuf(D3DK), kmem_zalloc(D3DK)

EXAMPLE
In a driver close(D2DK) routine, rather than clear each individual member of its
private data structure, the driver could use bzero as shown here:

bzero(&drv_dat[minor(dev)], sizeof(struct drvr_data));
NAME
canput - test for room in a message queue

SYNOPSIS
#include <sys/stream.h>
int canput(queue_t *cq);

ARGUMENT
   *cq    The pointer to the message queue. queue_t is an alias for the
           queue(D4DK) structure.

DESCRIPTION
canput searches through the stream (starting at cq) until it finds a queue containing
a service routine where the message can be enqueued, or until it reaches the
end of the stream. If found, the queue containing the service routine is tested to
see if there is room for a message in the queue. If the queue is full, canput sets
the QWANTW flag to back-enable the caller’s service routine.

NOTE: You are responsible for both testing a queue with canput and refraining
from placing a message on the queue if canput fails.

RETURN VALUE
If the message queue is not full, 1 is returned. A 0 is returned if the queue is
full.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
bcanput(D3DK), putbq(D3DK), putnext(D3DK)

EXAMPLE
See the bufcall(D3DK) function page for an example of canput.
NAME
clrbuf - erase the contents of a buffer

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
void clrbuf(struct buf *bp);

ARGUMENT
*bp
Pointer to the buf(D4DK) structure

DESCRIPTION
The clrbuf function zeros a buffer and sets the b_resid member of the buf structure to 0. Zeros are placed in the buffer starting at bp->b_un.b_words for a length of bp->b_bcount bytes. b_un.b_words and b_bcount are members of the buf structure defined in sys/buf.h.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
brelse(D3DK), buf(D4DK)

EXAMPLE
See biowait(D3DK).
NAME

`cmn_err` - display an error message or panic the system

SYNOPSIS

```c
#include <sys/cmn_err.h>

int cmn_err(int level, char *format, int args);
```

ARGUMENTS

- **level**
  - A constant defined in the `sys/cmn_err.h` header file. `level` indicates
    the severity of the error condition. The four severity levels are

  - **CE_CONT**
    - used to continue another message or to display an
      informative message not connected with an error.

  - **CE_NOTE**
    - used to display a message preceded with NOTICE. This
      message is used to report system events that do not
      necessarily require user action, but may interest the
      system administrator. For example, a message saying
      that a sector on a disk needs to be accessed repeatedly
      before it can be accessed correctly might be
      noteworthy.

  - **CE_WARN**
    - used to display a message preceded with WARNING. This
      message is used to report system events that require
      immediate attention, such as those where if an action
      is not taken, the system may panic. For example,
      when a peripheral device does not initialize
      correctly, this level should be used.

  - **CE_PANIC**
    - used to display a message preceded with PANIC or
      DOUBLE PANIC, and to panic the system. Drivers
      should specify this level only under the most severe
      conditions or when debugging a driver. A valid use of
      this level is when the system cannot continue to
      function. If the error is recoverable, or not essential to con-
      tinued system operation, do not panic the system. This
      level halts multiuser processing.

- **format**
  - The message to be displayed. By default, the message is sent both to
    the system console and to the kernel buffer `putbuf`. If the first character
    in `format` is an exclamation point (`'!'`), the message goes only to
    `putbuf`. If the first character in `format` is a circumflex (`'~'`), the message
    goes only to the console. Except for the first character, the rules for `format`
    are the same as those for `printf(3S)` strings. To read `putbuf`, use the following `crash(1M)` commands:

    ```
    od -d putbufsz
    od -a putbuf size
    ```

    The first command returns the size of `putbuf` (the default is 2000
    bytes). The second command uses the returned `size` to read `putbuf`. 
cmn_err appends \n to each format, even when a message is sent to putbuf, except when level is CE_CONT.

Valid conversion specifications are %s, %u, %d, %o, and %x. The cmn_err function is otherwise similar to the printf(3S) library subroutine in displaying messages on the system console or storing on putbuf.

NOTE: cmn_err does not accept length specifications in conversion specifications. For example, %3d is ignored.

args the set of arguments passed with the message being displayed. Any argument within the range of supported conversion specifications can be passed.

DESCRIPTION

cmn_err displays a specified message on the console and/or stores it in the putbuf array. cmn_err can also panic the system.

At times, a driver may encounter error conditions requiring the attention of a primary or secondary system console monitor. These conditions may mean halting multiuser processing; however, this must be done with caution. Except during the debugging stage, a driver should never stop the system.

The cmn_err function with the CE_CONT argument can be used by driver developers as a driver code debugging tool. However, using cmn_err in this capacity can change system timing characteristics.

If CE_PANIC is set, cmn_err stops the machine, after possibly a crash dump.

RETURN VALUE

None. However, if an unknown level is passed to cmn_err, the following panic error message is displayed:

```
PANIC: unknown level in cmn_err (level=level, msg=format)
```

LEVEL

Base or Interrupt

SEE ALSO

BCI Driver Development Guide, Chapter 12
print(D2DK), printf(3S)

EXAMPLE

The cmn_err function can record tracing and debugging information only in the putbuf (lines 15 and 16); display problems with a device only on the system console (line 21); or stop the system if a required device malfunctions (line 27).

```c
1 struct device { /* physical device registers layout */
2   int control;   /* physical device control word */
3   int status;    /* physical device status word */
4   int error;     /* error codes from device */
5   short recv_char; /* receive character from device */
6   short xmit_char; /* transmit character to device */
7 ; /* end device */
8
9 extern struct device xx_addr[]; /* physical device registers */
10 extern int xx_cnt; /* number of physical devices */
```
register struct device *rp;

rp = xx_addr[(getminor(dev) >> 4) & 0xf]; /* get dev registers */

#ifdef DEBUG
    /* in debugging mode, log function call */
    cmn_err(CE_NOTE, "xx_open function call, dev = 0x%"x", dev);
    cmn_err(CE_CONT, "! flag = 0x%"x", flag); /* continue msg */
#endif /* end DEBUG */

/* display device power failure on system console */

if ((rp->status & POWER) == OFF)
    cmn_err(CE_WARN, "xx_open: Power is OFF on device %d port %d",
            ((getminor(dev) >> 4) & 0xf), (getminor(dev) & 0xf));

/* halt system if root device has bad VTOC */
/* send message to system console and to putbuf */

if (rp->error == BADVTOC && dev == rootdev)
    cmn_err(CE_PANIC, "xx_open: Bad VTOC on root device");
NAME

copyb - copy a message block

SYNOPSIS

#include <sys/stream.h>

mblk_t *copyb(mblk_t *bp);

ARGUMENT

bp Pointer to the message block from which data is copied.

DESCRIPTION

copyb allocates a new message block, and copies into it the data from the block
pointed to by bp. The new block will be at least as large as the block being
copied. The b_rptr and b_wptr members of bp are used to determine how many
bytes to copy.

RETURN VALUE

If successful, copyb returns a pointer to the newly allocated message block con­
taining the copied data. Otherwise, it returns a NULL pointer.

LEVEL

Base or Interrupt

SEE ALSO

BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK)

EXAMPLE

For each message in the list, test to see if the downstream queue is full with the
canput(D3DK) function (line 21). If it is not full, use copyb(D3DK) to copy a
header message block, and dupmsg(D3DK) to duplicate the data to be retransmit­
ted. If either operation fails, reschedule a timeout at the next valid interval.

Update the new header block with the correct destination address (line 34), link
the message to it (line 35), and send it downstream (line 36). At the end of the
list, reschedule this routine.

1 struct retrns {
2     mblk_t *r_mp;
3     long r_address;
4     queue_t *r_outq;
5     struct retrns *r_next;
6 };
7
8 struct protoheader {
9     ...
10     long h_address;
11     ...
12 };
13
14 mblk_t *header;
15
16 retransmit(ret) register struct retrns *ret;


```c
16 {
17     register mblk_t *bp, *mp;
18     struct protoheader *php;
19
20     while (ret) {
21         if (!canput(ret->r_outq->q_next)) { /* no room */
22             ret = ret->r_next;
23             continue;
24         }
25         bp = copyb(header); /* copy header msg. block */
26         if (bp == NULL)
27             break;
28         mp = dupnsg(ret->r_mp); /* duplicate data */
29         if (mp == NULL) {
30             freeb(bp); /* free the block */
31             break;
32         }
33         php = (struct protoheader *)bp->b_rptr;
34         php->h_address = ret->r_address; /* new header */
35         bp->bp_cont = mp; /* link the message */
36         putnext(ret->r_outq, bp); /* send downstream */
37         ret = ret->r_next;
38     }
39     timeout(retransmit, (long)ret, RETRNS_TIME); /* reschedule */
40 }
```
NAME

-copyin - copy data from a user program to a driver buffer

SYNOPSIS

#include <sys/types.h>

int copyin(caddr_t userbuf, caddr_t driverbuf, int cn);

ARGUMENTS

userbuf  User program source address from which data is transferred.
driverbuf Driver destination address to which data is transferred.
cn       Number of bytes transferred.

DESCRIPTION

-copyin copies data from a user program source address to a driver buffer. The
driver developer must ensure that adequate space is allocated for the destination
address.

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

RETURN VALUE

Under normal conditions a 0 is returned indicating a successful copy. 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 area or stack area
- invalid address that would have resulted in data being copied into the
  user block

If a -1 is returned, returnEFAULT.

LEVEL

Base Only (Do not call from an interrupt routine)

SEE ALSO

bcopy(D3DK), copyout(D3DK), uiomove(D3DK)
NAME
copymsg - copy a message

SYNOPSIS
#include <sys/stream.h>

mblk_t *copymsg(mblk_t mp);

ARGUMENTS
mp Pointer to the message to be copied. mblk_t is an instance of the
msgb(D4DK) structure.

DESCRIPTION
copymsg forms a new message by allocating new message blocks, copies the con­
tents of the message referred to by mp (using the copyb(D3DK) function), and
returns a pointer to the new message.

RETURN VALUE
If the copy is successful, copymsg returns a pointer to the new message. Other­
wise, it returns a NULL pointer.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), copyb(D3DK), msgb(D4DK)

EXAMPLE
The routine lctouc converts all the lowercase ASCII characters in the message to
uppercase. If the reference count is greater than one (line 8), then the message is
shared, and must be copied before changing the contents of the data buffer. If
the call to the copymsg(D3DK) function fails (line 9), return NULL (line 10), other­
wise, free the original message (line 11). If the reference count was equal to 1,
the message can be modified. For each character (line 16) in each message block
(line 15), if it is a lowercase letter, convert it to an uppercase letter line 18). A
pointer to the converted message is returned (line 21).

```c
1 mblk_t *lctouc(mp)
2   mblk_t *mp;
3 {
4   mblk_t *cmp;
5   mblk_t *tmp;
6   unsigned char *cp;
7
8   if (mp->b_datap->db_ref > 1) {
9       if ((cmp = copymsg(mp)) == NULL)
10          return(NULL);
11       freemsg(mp);
12   } else {
13       cmp = mp;
14   }
15   for (tmp = cmp; tmp; tmp = tmp->b_next) {
16       for (cp = tmp->b_rptr; cp < tmp->b_wptr; cp++) {
```

17 if ((*cp <= 'z') && (*cp >= 'a'))
18    *cp -= 0x20;
19 }
20 }
21 return (cmp);
22 }
NAME

copyout - copy data from a driver to a user program

SYNOPSIS

#include <sys/types.h>

int copyout(caddr_t driverbuf, caddr_t userbuf, long en);

ARGUMENTS

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

userbuf  Destination address in the user program to which the data is transferred.

cn  Number of bytes moved.

DESCRIPTION

copyout copies data from driver buffers to user data space.

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

RETURN VALUE

Under normal conditions a 0 is returned to indicate a successful copy. Otherwise, a -1 is returned if the specified address range is not valid.

If a -1 is returned, returnEFAULT.

LEVEL

Base Only  (Do not call from an interrupt routine)

SEE ALSO


bcopy(D3DK), uiomove(D3DK), copyin(D3DK)

EXAMPLE

A driver ioctl(D2DK) routine (line 9) can be used to get or set device attributes or registers. In the XX_GETREGS condition (line 17), the driver copies the current device register values to a user data area (line 18). If the specified argument contains an invalid address, an error code is returned.

```c
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 }; /* end device */
7
8 extern struct device xx_addr[]; /* phys. device reg. location */
9 ...
10 xx_ioctl(dev, cmd, arg, flag)
11     dev_t dev;
12     caddr_t arg;
13     ...
14     {  register struct device *rp = &xx_addr[getminor(dev) >> 4];
```
switch(cmd) {
  case XX_GETREGS: /* copy device reqs. to user program */
    if (copyout((caddr_t)rp, arg, sizeof(struct device))
      return(EFAULT);
    /* endif */
    break;
}
NAME
datamsg - test whether a message is a data message

SYNOPSIS
#include <sys/stream.h>
#include <sys/ddi.h>

int datamsg(unsigned char type);

ARGUMENT
type The type of message to be tested. The db_type field of the clatab structure contains the message type. This field may be accessed through the message block using mp->b_datap->db_type.

DESCRIPTION
The datamsg function tests the type of message to determine if it is a data message type (M_DATA, M_DELAY, M_PROTO, or M_PCPROTO).

RETURN VALUE
datamsg returns 1 for TRUE, if the message is a data message; and 0 for FALSE for any other type of message.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
allocb(D3DK), clatab(D4DK), msgb(D4DK)

EXAMPLE
The put(D2DK) routine enqueues all data messages for handling by the srv(D2DK) (service) routine. All non-data messages are handled in the put routine.

```c
1   xxxput(q, mp)
2    queue_t *q;
3    mblk_t *mp;
4 {
5     if (datamsg(mp->b_datap->db_type)) {
6         putq(q, mp);
7         return;
8     }
9     switch (mp->b_datap->db_type) {
10         case M_FLUSH:
11             ...
12         }
```


NAME
delay - delay process execution for a specified number of clock ticks

SYNOPSIS
void delay(long ticks);

ARGUMENT
ticks The number of clock cycles for a delay. ticks are frequently set as an
expression containing the system variable HZ, the number of clock ticks
in one second; HZ is defined in sys/param.h.

DESCRIPTION
delay provides a way to wait for an event to happen. Occasionally, a driver may
need to wait a given period of time until work is available. The value of HZ can
vary from system to system, and so the function drv_hztousec(D3DK) should be
used when accurate timing is required.

The delay function calls timeout(D3DK) to schedule a wakeup call after the
specified amount of time has elapsed. delay then goes to sleep until timeout
wakes up the sleeping process. While delay is active, splhi is set. At comple­
tion, the former priority level is returned through splx.

delay requires user context.

RETURN VALUE
None

LEVEL
Base Only (Do not call from an interrupt routine)

SEE ALSO
BCI Driver Development Guide, Chapter 10, “Synchronizing Hardware and
Software Events”

biodone(D3DK), biowait(D3DK), drv_hztousec(D3DK), drv_usectohz(D3DK),
sleep(D3DK), timeout(D3DK), untimeout(D3DK), wakeup(D3DK)

EXAMPLE
Before a driver I/O routine allocates buffers and stores any user data in them, it
checks the status of the device (line 12). If the device needs manual intervention
(such as, needing to be refilled with paper), a message is displayed on the system
console (line 14). The driver waits an allotted time (line 16) before repeating the
procedure.

1 struct device { /* layout of physical device registers */
2       int control; /* physical device control word */
3       int status; /* physical device status word */
4 short xmit_char; /* transmit character to device */
5     }; /* end device */
6
7 extern struct device xx_addr[]; /* physical device regs. location */
     /* get device registers */
8
9 register struct device *rp = &xx_addr[getminor(dev)>>4]);
10
11 while(rp->status & NOPAPER) { /* while printer is out of paper */

13     /* display message and ring bell on system console */
14     cmn_err(CE_WARN, "Axx_write: NO PAPER in printer \d\007",
15     (dev & 0xf));
16     delay(60 * HZ);     /* wait one minute and try again */
17 } /* endwhile */
NAME
dcache_inval — invalidate the data cache

SYNOPSIS

include <sys/types.h>
include <sys/buf.h>
include <sys/iosystm.h>

void dcache_inval(paddr_t paddr int len);

ARGUMENTS

paddr the physical starting address
len the amount of memory to be invalidated

DESCRIPTION

If the standard driver support routines that automatically invalidate the cache (for
e.g., bp_iosetup) are not used, dcache_inval performs any explicit cache
invalidate operations that may be needed.

On systems that implement bus snooping, dcache_inval does nothing. On non-
snooping systems that cannot use the standard support routines, the data cache
must be invalidated before a read.

RETURN VALUE

None.

LEVEL

Base, interrupt.

SEE ALSO

dcache_sync(D3DK)
NAME
dcache_sync – sync the data cache

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>

void dcache_sync(paddr_t paddr int len);

ARGUMENTS
paddr the physical starting address
len the length to be synced

DESCRIPTION
If the standard driver support routines that automatically sync the cache (for
example, bp_iosetup) are not used, dcache_sync performs any explicit cache
sync operations that may be needed.

On systems that implement bus snooping, dcache_sync does nothing. On non­
snooping systems that cannot use the standard support routines, the data cache
must be synced before a write. On all systems, the data cache must be synced
following a read if any transferred data might be in the cache rather than in
memory (that is, if the driver copies data into the destination rather than using
DMA).

RETURN VALUE
None.

LEVEL
Base, interrupt.

SEE ALSO
dcache_inval(D3DK)
**NAME**

dma_pageio – break up an I/O request into manageable units

**SYNOPSIS**

```c
#include <sys/buf.h>

void dma_pageio(void (*strat) (), strat, struct buf *bp);
```

**ARGUMENTS**

- *strat* Pointer to the strategy(D2DK) routine to call to complete the I/O transfer.
- *bp* Pointer to the buf structure.

**DESCRIPTION**

dma_pageio breaks up a data transfer request from physiock(D3DK) into units of contiguous memory. The data is broken into 512-byte sectors until the last data bytes are encountered. dma_pageio executes spl0 around its internal sleep calls on reads and writes after the strategy routine is called. This may alter previously set spl(D3D) calls.

The driver must modify b_flags to indicate whether the transfer is a read or a write. OR in B_READ to indicate a read; turn B_READ off to indicate a write.

**RETURN VALUE**

None. However, conditions in dma_pageio can cause the following to be set:

- If memory for a temporary buffer cannot be allocated, b_flags is ORed with B_ERROR and B_DONE, and b_error is set to EAGAIN (resource temporarily unavailable). All allocated temporary buffers are deallocated when the transfer completes.

- If the I/O transfer is incomplete (b_flags does not contain B_DONE), then b_flags is set to B_WANTED and sleep(D3DK) is called to wait until a buffer can be allocated. The sleep priority is set to PRIVILEGED.

  The sleep code section is surrounded by a spl6-spl0 function set which may alter a previously set spl value.

- If B_ERROR is set after the strategy(D2DK) routine completes, allocated memory is freed and dma_pageio returns.

When the transfer completes, any allocated buffers are freed.

**LEVEL**

Base Only

**SEE ALSO**


**EXAMPLE**

The following example shows how dma_pageio is used when reading or writing disk data.

```c
struct dsize {
    daddr_t nblocks; /* number of blocks in disk partition */
    int cyloff;    /* starting cylinder # of partition */
} my_sizes[4] = {};
```
/* physical read */
my_read(dev, uio_p, cred_p)
    dev_t dev;
    uio_t *uio_p;
    cred_t *cred_p;
{
    register int nblks;
    /* get number of blocks in the partition */
    nblks = my_sizes[getminor(dev) & 0x7].nblocks;
    /* if request is within limits for the device, schedule I/O*/
    physiock(my_breakup, 0, dev, B_READ, nblks, uio_p);
}

/* physical write */
my_write(dev, uio_p, cred_p)
    dev_t dev;
    uio_t *uio_p;
    cred_t *cred_p;
{
    register int nblks;
    /* get the number of blocks in the partition */
    nblks = my_sizes[getminor(dev) & 0x7].nblocks;
    /* if request is within limits for the device, schedule I/O */
    physiock(my_breakup, 0, dev, B_WRITE, nblks, uio_p);
}

/* break up the request that came from physio into chunks of contiguous memory. Pass at least 512 bytes (one sector) at a time (except for the last request). */

static
my_breakup(bp)
    register struct buf *bp;
{
    dma_pageio(my_strategy, bp);
}
NAME
dma_sgio - break up an I/O request for controller that does scatter/gather

SYNOPSIS
#include <sys/buf.h>

void dma_sgio(void (*strat)() strat, struct buf *bp);

ARGUMENTS
*strat pointer to the strategy(D2DK) routine to call to complete the I/O transfer
bp pointer to the buf structure

DESCRIPTION
dma_sgio is used for physical I/O when the driver supports scatter/gather but the user transfer cannot be handled directly by the controller because of alignment problems.

dma_sgio allocates all the kernel virtual memory needed for the transaction (aligned on a page boundary), initiates the transaction, and copies the data to the user address space. If there is a limit to the number of blocks that can be DMAed by the controller, and the size of the read/write requests exceeds the limit, then the driver must invoke dma_sgio iteratively, without exceeding the maximum limit on each iteration, until the read/write request is satisfied.

If the user address is aligned suitably, the driver should initiate a DMA transaction directly to the user address by calling the strategy routine.

This routine is provided specifically for device driver support on the M68000 or M88000 family of processors.

RETURN VALUE
None.

LEVEL
Base only.

SEE ALSO
dma_pageio(D3DK)
NAME
drv_getparm — retrieve kernel state information

SYNOPSIS
#include <sys/ddi.h>

int drv_getparm(unsigned long parm, unsigned long *value_p);

ARGUMENTS
parm
The kernel parameter to be obtained from ddi.h. Possible values are

- LBOLT
  Read the value of the lbolt. (lbolt is an integer that represents the number of clock ticks since the last system reboot. This value is used as a counter or timer inside the system kernel.)

- PPGRP
  Read the process group identification number. This number determines which processes should receive a HANGUP or BREAK signal when detected by a driver.

- UPROCP
  Read the process table token value. This information is used for the second argument of the vtop(D3D) function.

- PPID
  Read process identification number.

- PSID
  Read process session identification number.

- TIME
  Read time in seconds.

- CPUBOARD
  Read the CPU board token. The values for the token may be found in the file /usr/include/sys/mvmecpu.h.

value_p
A pointer to the data space in which the value of the parameter is to be copied.

DESCRIPTION
This function verifies that parm corresponds to a kernel parameter that may be read. If the value of parm does not correspond to a parameter or corresponds to a parameter that may not be read, -1 is returned. Otherwise, the value of the parameter is stored in the data space pointed to by value_p.

drv_getparm does not explicitly check to see whether the device has the appropriate context when the function is called and the function does not check for correct alignment in the data space pointed to by value_p. It is the responsibility of the driver writer to use this function only when it is appropriate to do so and to correctly declare the data space needed by the driver.

RETURN VALUE
drv_getparm returns 0 to indicate success, -1 to indicate failure. The value stored in the space pointed to by value_p is the value of the parameter if 0 is returned, undefined if -1 is returned. -1 is returned if you specify a value other than LBOLT, PPGRP, PPID, PSID, TIME, CPUBOARD or UPROCP. Always check the return code when using this function.

LEVEL
Base only when using the PPGRP, PPID, PSID, TIME, or UPROCP argument values.
Interrupt usable when using the LBOLT and CPUBOARD argument value.
SEE ALSO
   vtop(D3D), buf(D4DK)
NAME
drv_hztousec - convert clock ticks to microseconds

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

clock_t drv_hztousec(clock_t hz);

ARGUMENT
hz The length of time (expressed in HZ units) to convert to its microsecond equivalent

DESCRIPTION
drv_hztousec converts into microseconds the length of time expressed by hz, which is in units of time based on the value of HZ, the kernel parameter whose value is defined in sys/param.h.

The kernel variable lbolt, which is readable through drv_getparm(D3DK), is the length of time the system has been up since boot and is expressed in HZ units. Drivers often use the value of lbolt before and after an I/O request to measure the amount of time it took the device to process the request. drv_hztousec can be used by the driver to convert the reading from HZ units, which could potentially vary between system implementations, to a known unit of time.

RETURN VALUE
The number of microseconds equivalent to the hz argument. No error value is returned. If the microsecond equivalent to hz is too large to be represented as a clock_t, then the maximum clock_t value will be returned.

LEVEL
Base or Interrupt

SEE ALSO
drv_getparm(D3DK), drv_usectohz(D3DK)
NAME
drv_prlv — determine driver privilege

SYNOPSIS
int drv_prlv(cred_t *cr);

ARGUMENT
*cr Pointer to the cred(D4DK) (credential) structure.

DESCRIPTION
The drv_prlv function provides a general interface to the system privilege policy.
It determines whether the credentials supplied by the cred structure pointed to
by cr identify a privileged process. This function should only be used when file
access modes and special minor device numbers are insufficient to provide pro­
tection for the requested driver function. It is intended to replace all calls to
suser() and any explicit checks for effective user ID = 0 in driver code.

RETURN VALUE
This routine returns 0 if it succeeds, EPERM if it fails.

LEVEL
Base or Interrupt

SEE ALSO
cred(D4DK)
NAME
drv_usectohz - convert microseconds to clock ticks

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

clock_t drv_usectohz(clock_t microsecs);

ARGUMENTS
microsecs  The number of microseconds to convert to its HZ equivalent.

DESCRIPTION
drv_usectohz converts a length of time expressed in microseconds to HZ, the
unit of time based on the the kernel parameter HZ whose value is defined in
sys/param.h. The time arguments to timeout(D3DK) and delay(D3DK) are
expressed in HZ, as well as the kernel variable lbolt, which is readable through
drv_getparm(LBOLT).

drv_usectohz is a portable way for drivers to make calls to timeout(D3DK) and
delay(D3DK) and remain binary compatible should the driver object file be made
part of a kernel that was compiled with a value of HZ different from that with
which the driver was compiled.

RETURN VALUE
The value returned is the number of HZ units equivalent to the microsecs argu-
ment. No error value is returned. If the HZ equivalent to microsecs is too large to
be represented as a clock_t, then the maximum clock_t value will be returned.

LEVEL
Base or Interrupt

SEE ALSO
drv_hztosec(D3DK)
Name

drv_usecwait - busy-wait for specified interval

Synopsis

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

void drv_usecwait(clock_t microsecs);

Argument

microsecs  The number of microseconds to busy-wait.

Description

The kernel function delay(D3DK) can be used by a driver to delay for a specified number of system ticks (given by parameter HZ in sys/param.h, which indicates how many system ticks occur per second). There are two limitations: (1) the granularity of the wait time is limited to 1/HZ second, which may be more time than is needed for the delay, and (2) delay(D3DK) may only be invoked with user context and hence cannot be used at interrupt time or system initialization.

Often, drivers need to delay for only a few microseconds, waiting for a write to a device register to be picked up by the device. In this case, even with user context, delay(D3DK) produces too long a wait period. The function drv_usecwait is provided to give drivers a means of busy-waiting for a specified microsecond count. The amount of time spent busy-waiting may be greater than the microsecond count but will minimally be the number of microseconds specified.

Note that the driver wastes processor time by making this call since drv_usecwait does not invoke sleep but simply busy-waits. The driver should only make calls to drv_usecwait as needed, and only for as much time as needed. drv_usecwait does not raise the processor interrupt level; if the driver wishes to mask out interrupts, it is its responsibility to set the priority level before the call and restore it to its original value afterward.

Return Value

None

Level

Base or Interrupt

See Also

delay(D3DK), timeout(D3DK), untimeout(D3DK)
NAME
dump – gain access to a device crash dump routine

SYNOPSIS
#include <sys/types.h>
iinclude <sys/file.h>
iinclude <sys/errno.h>
iinclude <sys/open.h>
iinclude <sys/cred.h>
iinclude <sys/crash.h>

int prefixdump(dev_t dev, int memsize, int *seqflag, int *startblk, int *blkcnt,
int *ptrblk, int *chunksize, int (**dumpfunc)());

ARGUMENTS
  dev 
    A device number.
  memsize
    Amount of physical memory to be dumped.
  seqflag
    Pointer to a flag which the driver sets to indicate whether the device is
    sequential or random access. The settings are defined in crash.h. 
    Valid settings are:
    CRASH_RAND_DEVICE the device is random access
    CRASH_SEQ_DEVICE the device is sequential access
  startblk
    Pointer which returns where the first block should be written for ran­
    dom access or returns 0 for sequential access.
  blkcnt
    Pointer which returns the number of blocks which should be written 
    to the device.
  ptrblk
    Pointer which returns where the pointer block should be written for 
    random access or returns 0 for sequential access.
  chunksize
    Pointer which returns the number of blocks the device can accept per 
    call to dumpfunc.
  dumpfunc
    Pointer which returns the address of the routine which will do the 
    writing for the crash dump. The routine's interface is as follows:
    int dumpfunc(dev_t dev, int blkno, paddr_t phys_addr, int cmd, int nblocks);
      dev 
        A device number.
      blkno
        The block number to write to.
      phys_addr
        The physical memory address to read from.
      cmd
        The command to perform, valid commands are:
        CRASH_DOIO_RD read from device
        CRASH_DOIO_WR write to device
        CRASH_DOIO_EOT do device specific action(s) at end of 
        crash dump
DESCRIPTION
The driver’s \texttt{dump} routine is called by the kernel through the \texttt{cdevsw} or \texttt{bdevsw} entry for the device during a panic via \texttt{cmn\_err(D3DK)}. The routine should verify that the minor number component of \texttt{dev} is valid, that dumping to this device is still valid, that the device is ready to perform a crash dump, and set the various parameters depending on whether it is accessed sequentially or by random access. If the device is a disk, only dumping to a partition tagged with V\_SWAP is allowed.

The routine returned by \texttt{dump} to do the I/O should transfer the data to the device and return when finished. If the device transfers by sequential access, this routine must also accept the command \texttt{CRASH\_DOIO\_EOT} to write the EOT mark(s) or take other device specific actions to finish the crash dump.

All blocks are specified as 512 bytes, the device is responsible for converting to the appropriate logical block size.

RETURN VALUE
The \texttt{dump} routine should return 0 for success, or -1 for failure.

SEE ALSO
\texttt{open(D2DK), close(D2DK), cmn\_err(D3DK)}
NAME
dupb - duplicate a message block descriptor

SYNOPSIS
#include <sys/stream.h>
mblk_t *dupb(mblk_t *bp);

ARGUMENTS
*bp Pointer to the message block to be duplicated. mblk_t is an instance of the msgb(D4DK) structure.

DESCRIPTION
dupb creates a new mblk_t structure to reference the message block pointed to by bp. Unlike copyb(D3DK), dupb does not copy the information in the data block, but creates a new structure to point to it.

The following figure shows how the db_ref field of the dblk_t structure has been changed from 1 to 2, reflecting the increase in the number of references to the data block. The new mblk_t contains the same information as the first. Note that b_rptr and b_wptr are copied from bp, and that db_ref is incremented.

![Figure showing before and after states of message block structure]

RETURN VALUE
If successful, dupb returns a pointer to the new message block. Otherwise, it returns a NULL pointer.

LEVEL
Base or Interrupt

SEE ALSO
copyb(D3DK)
EXAMPLE

This _srv(D3DK) (service) routine adds a header to all _M_DATA messages before passing them along. The message block for the header was allocated elsewhere. For each message on the queue, if it is a priority message, pass it along immediately (lines 9–10). Otherwise, if it is anything other than an _M_DATA message (line 11), and if it can be sent along (line 12), then do so (line 13). Otherwise, put the message back on the queue and return (lines 15–16). For all _M_DATA messages, first check to see if the stream is flow-controlled (line 19). If it is, put the message back on the queue and return (line 22); if it is not, the header block is duplicated (line 20). If _dupb fails, the service routine is rescheduled in one tenth of a second (HZ/10) with timeout and then we return (lines 23–24). If _dupb succeeds, link the _M_DATA message to it (line 26) and pass it along (line 27). _dupb can be used here instead of _copyb(D3DK) because the contents of the header block are not changed.

```c
1 void srv(q)
2     queue_t *q;
3 {
4     mblk_t *mp;
5     mblk_t *bp;
6     extern mblk_t *hdr;
7
8     while ((mp = getq(q)) != NULL) {
9         if (mp->b_datap->db_type >= QPCTL) {
10             putnext(q, mp);
11         } else if (mp->b_datap->db_type != _M_DATA) {
12             if (canput(q->q_next))
13                 putnext(q, mp);
14             else {
15                 putbq(q, mp);
16                 return;
17             }
18         } else { /* _M_DATA */
19             if (canput(q->q_next)) {
20                 bp = _dupb(hdr);
21                 if (bp == NULL) {
22                     putbq(q, mp);
23                     timeout(qenable, (long)q, HZ/10);
24                     return;
25                 }
26                 linkb(bp, mp);
27                 putnext(q, bp);
28             } else {
29                 putbq(q, mp);
30                 return;
31             }
32         }
33     }
34 }
```
NAME
dupmsg - duplicate a message

SYNOPSIS
#include <sys/stream.h>
mblk_t *dupmsg(mblk_t *mp);

ARGUMENTS
mp Pointer to the message block.

DESCRIPTION
dupmsg forms a new message by copying the message block descriptors pointed to by mp and linking them. dupb(D3DK) is called for each message block. The data blocks themselves are not duplicated.

RETURN VALUE
If successful, dupmsg returns a pointer to the new message block. Otherwise, it returns a NULL pointer.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
copyb(D3DK), copymsg(D3DK), dupb(D3DK)

EXAMPLE
See the copyb(D3DK) function page for an example of dupmsg.
enableok (D3DK)  DDI/DKI (STREAMS)  enableok (D3DK)

NAME
enableok - reschedule a queue for service

SYNOPSIS
#include <sys/stream.h>
#include <sys/ddi.h>

void enableok(queue_t *q);

ARGUMENT
q A pointer to the queue to be rescheduled.

DESCRIPTION
The enableok function allows queue q to be rescheduled for service. It cancels
the effect of a previous use of the noenable(D3DK) function on q by turning off
the QNOENB flag in the queue.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
noenable(D3DK), qenable(D3DK)

EXAMPLE
The qrestart routine uses two STREAMS functions to restart a queue that has
been disabled. The enableok function turns off the QNOENB flag, allowing the
qenable(D3DK) function to schedule the queue for immediate processing.

```c
void qrestart(rdwr q)
{
    register queue_t *rdwr_q;

    enableok(rdwr_q);
    /* re-enable a queue that has been disabled */
    (void) qenable(rdwr_q);
}
```
esballoc (D3DK)  

NAME
esballoc - allocate a message block using a shared buffer

SYNOPSIS
#include <sys/stream.h>

mblk_t *esballoc(unsigned char *base, int size, int pri, 
frtn_t *fr_rtnp);

ARGUMENTS
base Address of user supplied data buffer.
size Number of bytes in data buffer.
pri Priority of allocation request (to be used by allocb(D3DK) function, 
called by esballoc).
fr_rtnp Free routine data structure.

DESCRIPTION
esballoc creates a STREAMS message and attaches a user-supplied data buffer 
in place of a STREAMS data buffer. It calls allocb(D3DK) to get a message and 
data block header only. The user-supplied data buffer, pointed to by base, is used 
as the data buffer for the message.

The fr_rtn structure is referenced by the dp_freep member of the clatab 
structure. When freeb(D3DK) is called to free the message, the driver's message 
freeing routine (referenced through the fr_rtn structure) is called, with argu­ 
ments, to free the data buffer.

The fr_rtn structure has the following declaration:

    struct free_rtn {
        void (*free_func)(); /* user's freeing routine */
        char *free_arg; /* arguments to free_func() */
    }

typedef struct free_rtn frtn_t;

Instead of requiring a specific number of arguments, the free_arg field is 
defined of type char *. This way, the driver can pass a pointer to a structure if 
more than one argument is needed.

NOTE: The free_func function must be defined in kernel space, should be 
declared void and accept one argument. It has no user context and must not 
sleep.

RETURN VALUE
On success, a pointer to the newly allocated message block is returned. On 
failure, NULL is returned.

LEVEL
Base or Interrupt

SEE ALSO
allocb(D3DK), freeb(D3DK), clatab(D4DK), free_rtn(D4DK)
esbbcall(D3DK)  DDI/DKI(STREAMS)  esbbcall(D3DK)

NAME
esbbcall - call function when buffer is available

SYNOPSIS
#include <sys/stream.h>

mblk_t *esbbcall(int pri, int func, long arg);

ARGUMENTS
  pri    Priority of allocation request (to be used by allocb(D3DK) function,
calling esbbcall)
  func   Function to be called when buffer becomes available.
  arg    Argument to func.

DESCRIPTION
esbbcall, like bufcall(D3DK), serves as a timeout(D3DK) call of indeterminate
length. If esballoc(D3DK) is unable to allocate a message and data block header
to go with its externally supplied data buffer, esbbcall can be used to schedule
the routine func, to be called with the argument arg when a buffer becomes avail­
able. func may be a routine that calls esbbcall or it may be another kernel func­tion.

RETURN VALUE
On success, 1 is returned. On failure, 0 is returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), bufcall(D3DK), datab(D4DK), esballoc(D3DK)
etomajor - convert external to internal major device number

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

int etomajor(major_t emaj);

ARGUMENT
        emaj       An external major number.

DESCRIPTION
        etomajor converts the external major number (emaj) to an internal major number.

RETURN VALUE
        etomajor returns the internal major number or NODEV if the external major number exceeds the bdevsw and cdevsw count.

LEVEL
        Base or Interrupt

SEE ALSO
        getemajor(D3D), getemajor(D3D), getmajor(D3DK), getminor(D3DK),
        itoemajor(D3D), makedevice(D3DK)
flushband (D3DK)  DDI/DKI(STREAMS)  flushband (D3DK)

NAME
flushband - flush messages for a specified priority band

SYNOPSIS
#include <sys/stream.h>
void flushband(queue_t q, unsigned char pri, int flag);

ARGUMENTS
q Pointer to the queue.
pri Priority of messages to be flushed.
flag Valid flag values are:

  FLUSHDATA  Flush only data messages (types M_DATA, M_DELAY, M_PROTO, and M_PCPROTO).
  FLUSHALL   Flush all messages.

DESCRIPTION
The flushband function flushes messages associated with the priority band specified by pri. If pri is 0, only normal and high priority messages are flushed. Otherwise, messages are flushed from the band pri according to the value of flag.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
flushq(D3DK)
NAME
flushq - remove messages from a queue

SYNOPSIS
#include <sys/stream.h>
void flushq(queue_t *q, int flag);

ARGUMENTS
*q Pointer to the queue to be flushed.
flag Valid flag values are:
   FLUSHDATA Flush only data messages (types M_DATA, M_DELAY,
                 M_PROTO, and M_PCPROTO).
   FLUSHALL  Flush all messages.

DESCRIPTION
flushq frees messages and their associated data structures by calling
freemsg(D3DK). If the queue's count falls below the low water mark and
QWANTW is set, the nearest upstream service procedure is enabled.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
freemsg(D3DK), putq(D3DK)

EXAMPLE
This example depicts the canonical flushing code for STREAMS modules. The
module has a write service procedure and potentially has messages on the queue.
If it receives an M_FLUSH message, and if the FLUSHR bit is on in the first byte of
the message (line 10), then the read queue is flushed (line 11). If the FLUSHW bit is
on (line 12), then the write queue is flushed (line 13). Then the message is passed
along to the next entity in the stream (line 14). See the example for
qreply(D3DK) for the canonical flushing code for drivers.

1 /*
2 * Module write-side put procedure.
3 */
4 xxxwput(q, mp)
5 queue_t *q;
6 mblk_t *mp;
7 {
8     switch(mp->b_datap->db_type) {
9         case M_FLUSH:
10             if (*mp->b_rptr & FLUSHR)
11                 flushq(RD(q), FLUSHALL);
12             if (*mp->b_rptr & FLUSHW)
13                 flushq(q, FLUSHALL);
14             putnext(q, mp);
flushq (D3DK)  

DDI/DKI(STREAMS)  

flushq (D3DK)  

15      break;
16      
17      

...
freeb (D3DK)  DDI/DKI (STREAMS)  freeb (D3DK)

NAME
freeb - free a message block

SYNOPSIS
#include <sys/stream.h>

void freeb(mblk_t *bp);

ARGUMENTS
bp Pointer to the message block to be deallocated. mblk_t is an instance
of the msgb (D4DK) structure.

DESCRIPTION
freeb deallocates a message block. If the reference count of the db_ref member
of the datab (D4DK) structure is greater than 1, freeb decrements the count. If
db_ref equals 1, it deallocates the message block and the corresponding data
block and buffer.

If the data buffer to be freed was allocated with the esballoc (D3DK) function,
the buffer may be a non-STREAMS resource. In that case, the driver must be
notified that the attached data buffer needs to be freed, and run its own freeing
routine. To make this process independent of the driver used in the stream,
freeb finds the free_rtn (D4DK) structure associated with the buffer. The
free_rtn (D4DK) structure contains a pointer to the driver-dependent routine,
which releases the buffer. Once this is accomplished, freeb releases the
STREAMS resources associated with the buffer.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), dupb(D3DK), esballoc(D3DK), free_rtn(D4DK)

EXAMPLE
See the copyb (D3DK) function page for an example of freeb.
NAME
freemsg – free all message blocks in a message

SYNOPSIS
#include <sys/stream.h>

int freemsg(mblk_t *mp);

ARGUMENT
mp Pointer to the message blocks to be deallocated. mblk_t is an instance of the msgb(D4DK) structure.

DESCRIPTION
freemsg calls freeb(D3DK) to free all message and data blocks associated with the message pointed to by mp.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
freeb(D3DK)

EXAMPLE
See the copymsg(D3DK) function page for an example of freemsg.
freerbuf(D3DK)  DDI/DKI  freerbuf(D3DK)

NAME
freerbuf — free a raw buffer header

SYNOPSIS
#include <sys/buf.h>
#include <sys/ddi.h>

void freerbuf(struct buf *bp);

ARGUMENTS
*bp Pointer to a previously allocated buffer header structure.

DESCRIPTION
freerbuf frees a raw buffer header previously allocated by getrbuf(D3DK).
This function does not sleep and so may be called from an interrupt routine.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
getrbuf(D3DK), kmem_alloc(D3DK), kmem_free(D3DK), kmem_zalloc(D3DK)
NAME
getemajor - get external major device number

SYNOPSIS
#include <sys/types.h>
#include <sys/ddi.h>
major_t getemajor(dev_t dev);

ARGUMENT
    dev  An external device number (contains both the major and minor number).

DESCRIPTION
getemajor returns the external major number given a device number, dev.

RETURN VALUE
The external major number.

LEVEL
Base or Interrupt

SEE ALSO
geteminor(D3D), etoimajor(D3D), getmajor(D3DK), makedevice(D3DK),
getminor(D3DK)
NAME
geteminor - get external minor device number

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

minor_t geteminor(dev_t dev);

ARGUMENT
dev
External device number.

DESCRIPTION
geteminor returns the external minor number given a device number, dev.

RETURN VALUE
The external minor number.

LEVEL
Base or Interrupt

SEE ALSO
getemajor(D3D), etoimajor(D3D), getmajor(D3DK), makedevice(D3DK), getminor(D3DK)
NAME
geterror - return I/O error

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

int geterror(struct buf *bp);

ARGUMENT
bp        Pointer to the block interface buffer structure defined in buf.h.

DESCRIPTION
geterror is called to retrieve the error number from the error field of the buffer
header structure.

RETURN VALUE
An error number indicating the error condition of the I/O request is returned. If
the I/O requested is completed successfully, 0 is returned.

LEVEL
Base or Interrupt

SEE ALSO
buf(D4DK)
NAME
getmajor - get major or internal major device number

SYNOPSIS
#include <sys/types.h>
#include <sys/mkdev.h>
#include <sys/ddi.h>

major_t getmajor(dev_t dev);

ARGUMENT
dev Device number.

DESCRIPTION
The getmajor function extracts either the major number or the internal major
number from a device number. For the MC88000 and MC68000 architectures,
getmajor returns the internal major number. For architectures that do not make
a distinction between internal and external major numbers, getmajor returns the
major number.

RETURN VALUE
The major number or internal major number.
NOTE: No validity checking is performed. If dev is invalid, an invalid number is
returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 3, "Drivers in the UNIX Operating System"
makedevice(D3DK), getminor(D3DK)
NAME
getminor — get minor or internal minor device number

SYNOPSIS
#include <sys/types.h>
#include <sys/mkdev.h>
#include <sys/ddi.h>

minor_t getminor(dev_t dev);

ARGUMENT
    dev       Device number.

DESCRIPTION
The getminor function extracts either the minor number or the internal minor
number from a device number. For the MC88000 and MC68000 architectures,
getminor returns the internal minor number. For architectures that do not make
a distinction between internal and external minor numbers, getminor returns the
minor number.

RETURN VALUE
The minor number or internal minor number.

NOTE: No validity checking is performed. If dev is invalid, an invalid number is
returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 3, “Drivers in the UNIX Operating System”
getmajor(D3DK), makedevice(D3DK)
getq(D3DK) DDI/ DKI (STREAMS) getq(D3DK)

NAME
getq - get the next message from a queue

SYNOPSIS
#include <sys/stream.h>
mblk_t *getq(queue_t *q);

ARGUMENTS
q Pointer to the queue from which the message is to be retrieved.

DESCRIPTION
getq is used by a service (srv(D2DK)) routine to retrieve its enqueued messages. A module or driver may include a service routine to process enqueued messages. Once the STREAMS scheduler calls srv it must process all enqueued messages, unless prevented by flow control. getq gets the next available message from the top of the queue pointed to by q. It should be called in a while loop that should be exited only when there are no more messages. getq turns the QWANTR flag off when a queue is being read, and turns QWANTR on when there are no more messages. When QWANTW is set it means an attempt has been made to write to the queue while it was blocked by flow control. If this is the case, getq back-enables (restarts) the service routine once it falls below the low water mark.

RETURN VALUE
If there is a message to retrieve, getq returns a pointer to it. If no message is queued, getq returns a NULL pointer.

LEVEL Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
Programmer's Guide: STREAMS, Chapter 5, "Messages"
bcanput(D3DK), canput(D3DK), putbq(D3DK), putq(D3DK), qenable(D3DK), srv(D2DK)

EXAMPLE
See the dupb(D3DK) function page for an example of getq.
NAME
getrbuf - get a raw buffer header

SYNOPSIS
#include <sys/buf.h>
#include <sys/kmem.h>
#include <sys/ddi.h>

struct buf *getrbuf(long sleepflag);

ARGUMENT

sleepflag Indicates whether driver should sleep for free space.

DESCRIPTION

getrbuf allocates the space for a buffer header to the caller. It is used in cases
where a block driver is performing raw (character interface) I/O and needs to set
up a buffer header that is not associated with the buffer cache.

getrbuf calls kmem_alloc(D3DK) to perform the memory allocation.
kmem_alloc requires the information included in the sleepflag argument. If
sleepflag is set to KM_SLEEP, the driver may sleep until the space is freed up. If
sleepflag is set to KM_NOSLEEP, the driver will not sleep. In either case, a pointer
to the allocated space is returned or NULL to indicate that no space was available.

RETURN VALUE

A pointer to the allocated buffer header, or NULL if no space is available.

LEVEL

Base or Interrupt (must not sleep if calling from interrupt routine)

SEE ALSO

freerbuf(D3DK), kmem_alloc(D3DK), kmem_free(D3DK)
NAME
hat_getkpfnnum – get page frame number for kernel address

SYNOPSIS
#include <sys/vrn.h>
#include <sys/types.h>

u_int hat_getkpfnnum(caddr_t addr);

ARGUMENT
addr The kernel virtual address for which the page frame number is to be returned.

DESCRIPTION
Drivers implementing the mmap(D2K) entry point must return -1 (for error) or the page frame number corresponding to the virtual address of the device memory addr. This frame number can be obtained by a call to hat_getkpfnnum.

RETURN VALUE
The page frame number corresponding to virtual address addr. There is no special error return value; invalid addresses will produce meaningless return values.

LEVEL
Base or interrupt. Although there is no reason why hat_getkpfnnum cannot be called at interrupt level, there is no need since it only needs to be called from mmap(D2K).

SEE ALSO
mmap(D2K), page_numtopp(D2DK), page_pptonum(D2DK)
NAME
hdeeqd - initialize hard disk error logging

SYNOPSIS
#include <sys/types.h>
#include <sys/hdelog.h>
#include <sys/mkdev.h>

int hdeeqd(dev_t dev, daddr_t pdsno, short edtyp);

ARGUMENTS

  dev     External device number (contains both the major number and the minor number). The driver must call the expdev macro (defined in sysmacros.h) to compress the device number.
  pdsno   Physical description sector
  edtyp   Error device type. The valid values are
           EQD_EFC  external floppy controller
           EQD_EHDC external hard disk controller
           EQD_ID   integral disk drive
           EQD_IF   integral floppy disk drive
           EQD_TAPE cartridge tape drive

DESCRIPTION
hdeeqd initializes information in the hard disk error logging table for the device specified by dev. This function is called once per device.

NOTE: This function is not part of the default set of kernel functions. Ensure that the HDE bootable object module is placed in the /boot directory.

RETURN VALUE
Under all conditions, a 0 is returned. However, internal errors can occur in hdeeqd causing a warning message to display on the console. Errors occur in the following conditions:

  The internal major device number is greater than or equal to the number of the controllers, called cdevcnt, which is assigned by /boot when the operating system is loaded. The message is

    WARNING: hdeeqd: major(ddev) = int-major (>=cdevcnt)

    int-major is the internal major device number.

  The count of used disk slots in the error logging table exceeds the number of available slots. The message is

    WARNING: Too few HDE equipped slots
    bad block handling skipped for maj/min = ext-maj, ext-min

    ext-maj and ext-min are the external major and minor numbers.

LEVEL
Base or Interrupt
SEE ALSO
BCI Driver Development Guide, Chapter 12, “Error Reporting”
hdelog(D3D), hdedata(D40)

EXAMPLE

When a device is opened for the first time, the driver open(D2DK) or init(D2D) routines (open in this example) must identify the device and set up controlling information about the device. In this example, the information is kept on a controlling sector on the disk. If the controlling sector does not exist, the information is encoded as a static table in the driver.

```c
#define XX_CNTL_BLKNO 0 /* controlling sector block number */
struct device {
    /* physical device registers layout */
    char reserve[4]; /* reserve space on card */
    ushort control; /* physical device control word */
    char status; /* physical device status word */
    char ivec_num; /* device interrupt vector number in */
    /* 0xF0; subdevice reporting in 0x00 */
    paddr_t addr; /* data address to be read/written */
    int count; /* amount of data to be read/written */
}; /* end device */

}; /* end xx_ */

struct xx_info {
    /* information on control sector */
    long xx_id; /* disk device id code */
    long xx_cyl; /* total number of cylinders */
    long xx_trk; /* number of tracks per cylinder */
    long xx_sec; /* number of sectors per track */
    char xx_serial[12]; /* device serial number */
}; /* end xx_info */

extern struct xx_ xx_devtab[]; /* logical device structures table */
extern struct device *xx_addr[]; /* physical dev registers location */
extern struct xx_info xx_info[]; /* device control information */
extern int xx_cnt; /* number of devices */

xx_open(dev, flag, otyp, crp)
    dev_t *dev;
    int flag, otype;
    struct cred *crp;
{
    register struct xx_ *dp;
    register struct device *rp;
```
register int unit;

... 

unit = getminor(dev) >> 4;  /* get drive unit number */
dp = &xx_devtab[unit];    /* get logical device information */
if ((dp->xx_flag & XX_OPEN) == 0) {  /* if first time device opened */
    hdeeqd(dev, XX_CNTLBLKNO, EQD_ID);  /* initialize error logging */
    bp = kmem_alloc(1024, KM_NOSLEEP);  /* get control sector buffer */
    bp->b_flags = B_READ;  /* set up buffer to read */
    bp->b_blkno = XX_CNTLBLKNO;  /* control sector from disk */
    bp->b_count = 512;
    bp->b_e_dev = dev;
    xx_strategy(bp);  /* read control sector */
    biowait(bp);  /* wait for read to complete */
    if ((bp->b_flags & B_ERROR) != 0) {
        /* if data error occurred, display message on console */
        xx_print(dev, "xx_open: cannot read control sector");
    } else {
        /* copy control sector data to info table */
        bcopy(bp->b_un.b_addr, &xx_info[unit], sizeof(struct xx_info));
        hdeeqd(dev, XX_CNTLBLKNO, EQD_ID);  /* start error logging */
        dp->flag |= XX_OPEN;  /* indicate device open */
    }
    brelse(bp);  /* release system buffer */
} /* endif */

If this is the first open, hdeeqd (line 44) is used to initiate error logging for the device. A system buffer is allocated (line 45) and the driver reads the controlling sector from the xx_strategy routine (line 50). If an error occurred on the read attempt, an error message is displayed (line 54) and an error condition is returned. Otherwise, the driver saves information from the controlling sector with bcopy (line 56) and indicates the device has been opened. Finally, the system buffer is released (line 60).
NAME
hdelog — log hard disk error

SYNOPSIS
#include <sys/types.h>
#include <sys/hdelog.h>
#include <sys/mkdev.h>

int hdelog(struct hdedata *eptr);

ARGUMENT
eptr  Pointer to the hdedata(D4D) structure defined in sys/hdelog.h. The
driver developer places information in the structure before hdelog is
called.

DESCRIPTION
hdelog logs a hard disk error in the error logging queue and displays a warning
message on the console to alert the operator to the problem.

The console message is
WARNING:  severity readtype hard disk error:
maj/min = external-major-num, external-minor-num

where severity is “marginal” or “unreadable”, and readtype is “CRC” (cyclic
redundancy check) or “ECC” (error check and correction).

hdeeqd(D3D) must be called once before this function to initialize error logging.
hdelog logs disk drive media errors. NOTE: This function is not part of the
default kernel. Ensure that the HDE bootable object module is placed in the /boot
directory.

Before calling this function, values must be assigned to the hdedata(D4D) struc­
ture. These members include the expanded device number; the disk pack serial
number; the physical block address; the type of read operation CRC or ECC;
whether the error is marginal or whether the disk is unreadable; the number of
unreadable tries; the bit width of the corrected error; and a time stamp.

RETURN VALUE
Under all conditions, a 0 is returned. However, an internal error can occur in hde­
log causing a warning message to display on the console. This error occurs
when the error logging table is full. In this case, the usual disk error warning
message is prefaced with
WARNING:  HDE queue full, following report not logged

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 12, “Error Reporting”
hdeeqd(D3D), hdedata(D4D)

EXAMPLE
A driver interrupt routine must check for data transfer errors (called data checks).
When a data check occurs (reported by the device in the status or error register),
the driver determines if there have been sufficient attempts to resolve the error.
If so, the driver abandons the I/O request by marking the buffer as being in error, logging an unresolved error (line 60), and marking the I/O operation complete (line 61). When an error persists in spite of multiple attempts to resolve it, the driver logs marginal errors (line 75) and attempts the I/O operation again. The driver may try to resolve the error with software by using the error correction bits in an error check and correction (ECC) register. See hdedata(D4D) for a description of the xx_edata structure shown in this example line 17.

```c
struct device {
  char reserve[4]; /* reserve space on card */
  ushort control; /* physical device control word */
  char status; /* physical device status word */
  char ivec_num; /* device interrupt vector no. in */
  paddr_t addr; /* address of data read/written */
  int count; /* amount of data read/written */
}; /* end device */

struct xx_ {
  struct buf *xx_head; /* I/O buffer queue head pointer */
  struct buf *xx_tail; /* I/O buffer queue tail pointer */
  short xx_flag; /* logical status flag */
  struct hdedata xx_edata; /* hard disk error record */
  struct iostat xx_stat; /* unit I/O stats for setting an */
  /* error rate during error logging */
}; /* end xx_ */

struct xx_info {
  long xx_id; /* device id code */
  long xx_cyl; /* total number of cylinders */
  long xx_trk; /* number of tracks per cylinder */
  long xx_sec; /* number of sectors per track */
  char xx_serial[12]; /* device serial number */
}; /* end xx_info */

extern struct xx xx_devtab[]; /* logical dev structures table */
extern struct device *xx_addr[]; /* physical dev register location */
extern struct xx_info xx_info[]; /* device control information */
extern int xx_int(int board)
  int board;
  /* get dev registers */
  register struct device *rp = xx_addr[board];
  register struct xx_ *dp;
  register struct buf *bp;
  register int  unit;

  unit = (board << 4) | (rp->ivec_num & 0xf); /* make unit number */
dp = &xx_devtab[unit];
if ((rp->status & DATACHK) != 0) {
    /* if data check error occurred */
    if (++dp->xx_edata.badrtcnt > XX_MAXTRY) { /* if sufficient */
        /* attempts have been made, then abandon the I/O request */
        bp = dp->xx_head; /* get buffer from I/O queue */
        dp->xx_head = bp->av_forw; /* remove buffer from I/O queue */
        bp->b_flags |= B_ERROR; /* mark buffer as being in error */
        bp->b_error = EIO; /* supply error condition */
        /* supply information needed for error logging */
        dp->xx_edata.diskdev = bp->b_edev; /* device number */
        dp->xx_edata.blkaddr = bp->b_blkno; /* block no. in error */
        dp->xx_edata.readtype = HDEECC; /* error type: error check */
        dp->xx_edata.severity = HDEUNRD; /* data was unreadable */
        dp->xx_edata.bitwidth = 0;
        dp->xx_edata.timestmp = time; /* time recording occurred */
        bcopy(dp->xx_edata.dskserno, xx_info[unit].serial, 12);
        hdelog(&dp->xx_edata); /* log abandoned I/O operations*/
        biodone(bp); /* mark I/O operation complete */
    } else if(dp->xx_edata.badrtcnt > 1) { /* if more than one retry */
        /* log error as marginal */
        bp = dp->xx_head; /* get buffer from I/O queue but leave on */
        /* I/O queue so that I/O operation is repeated */
        /* supply information needed for error logging */
        dp->xx_edata.diskdev = bp->b_edev; /* device number */
        dp->xx_edata.blkaddr = bp->b_blkno; /* error block number */
        dp->xx_edata.readtype = HDEECC; /* err. type: error check */
        dp->xx_edata.severity = HDEMARG; /* marginal error */
        dp->xx_edata.bitwidth = 0;
        dp->xx_edata.timestmp = time; /* time recording occurred */
        bcopy(dp->xx_edata.dskserno, xx_info[unit].serial, 12);
        hdelog(&dp->xx_edata); /* log data check error */
    } /* endif */
} /* endif */
NAME
  iomapin - map an I/O address (device)

SYNOPSIS
  #include <sys/types.h>
  #include <sys/buf.h>
  #include <sys/iosystm.h>
  
  int iomapin(unsigned int bus, addr_t addr, int len);

ARGUMENTS
  bus       I/O bus where device resides.
  addr      Address on bus.
  len       Amount to be mapped in.

DESCRIPTION
  In UNIX System V/88, if a device is present, it can be mapped in via iomapin.
  bus is the I/O bus where the device resides (for example, VME_A16 or VME_A32).
  iomapin may use any leftover data BATCs to map the area 1-1, or it may use
  seg_kmem to map the device into the kernel virtual segment.
  In , devices are mapped in 1-1 and iomapin simply returns the address (thus, it's
  essentially a no-op).

RETURN VALUE
  The address used to access the device mapped in.

LEVEL
  Base

SEE ALSO
  ioprobe(D3DK)
NAME
iomem_alloc - allocate physically contiguous memory

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>

int iomem_alloc(int nbytes, int flags);

ARGUMENTS
nbytes  size, in bytes, of the request
flags   special processing request

DESCRIPTION
The memory returned is physically contiguous and virtually mapped (no guaran­
tee of 1:1 mapping).

iomem_alloc accepts the flags IOM_NOSLEEP and IOM_NOCACHE. The
IOM_NOSLEEP flag should be set if the requester will tolerate waiting for the
request memory. The IOM_NOCACHE flag should be set if the requested pages
should be marked as cache-inhibited after they have been successfully allocated.
Because iomem_alloc fails (returning NULL) when the requested number of con­
tiguous pages cannot be found, it should be called at system initialization only.

NOTE: Memory allocated by iomem_alloc is not paged. Available memory is
therefore limited. Excessive use of this memory is likely to affect overall system
performance.

RETURN VALUE
If successful, iomem_alloc returns the address of the first byte of the contiguous
memory allocated. On UNIX System V/88, NULL is returned if IOM_NOSLEEP is
set and memory cannot be allocated.

LEVEL
Base.

SEE ALSO
iomem_free(D3DK), kmem_alloc(D3DK)
NAME
iomem_free - free memory allocated by iomem_alloc

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>

int iomem_free(caddr_t *addr, int nbytes);

ARGUMENTS
*addr pointer to the virtual address
nbytes size, in bytes, of resource released

DESCRIPTION
iomem_free is used to free memory allocated by iomem_alloc. Repetitious use of alloc/free operations is not advised.

LEVEL
Base.

SEE ALSO
iomem_alloc(D3DK), kmem_free(D3DK)
NAME
ioprobe – probe an I/O address for a device

SYNOPSIS
UNIX System V/68:
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>
int ioprobe(u_int access, addr_t addr, u_int *data);

UNIX System V/88:
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>
int ioprobe(u_int bus, addr_t addr);

ARGUMENTS
bus The I/O bus to probe (UNIX System V/88 only).
access The type of probe desired (UNIX System V/68 only).
addr The Address to probe.
data The data read or written by ioprobe (UNIX System V/68 only).

DESCRIPTION
ioprobe is used in driver initialization routines to probe for a controller at addr.

On UNIX System V/68, the controller may be probed in different ways depending on the value of access. The following constants are used to define the access type:

IOP_READ 0
IOP_WRITE 1
IOP_BYTE 2
IOP_SHORT 4
IOP_LONG 6
IOP_DEFBUS 0

ioprobe is called with an access value that is a combination (bit-wise “or”) of these constants. Acceptable access values (defined in sys/iosystm.h) and the resulting behavior of ioprobe follow:

IOP_READ|IOP_BYTE|IOP_DEFBUS Read a byte from addr.
IOP_READ|IOP_SHORT|IOP_DEFBUS Read an unsigned short integer from addr.
IOP_READ|IOP_LONG|IOP_DEFBUS Read an unsigned long integer from addr.
IOP_WRITE|IOP_BYTE|IOP_DEFBUS Write a byte to addr.
IOP_WRITE|IOP_SHORT|IOP_DEFBUS Write a short to addr.
IOP_WRITE|IOP_LONG|IOP_DEFBUS Write a long to addr.

On UNIX System V/88, a specific bus may be selected when ioprobe is called. bus (defined in sys/iom system.h) may be one of the following:

VME_A16 0x4
VME_A24 0x5
VME_A32 0x6

If addr is invalid or nonexistent, ioprobe returns -1.

RETURN VALUE
The address of the device, or -1 if no device responded.

LEVEL
Base

SEE ALSO
iomapin(D3DK)
NAME

insq - insert a message into a queue

SYNOPSIS

#include <sys/stream.h>

int insq(queue_t *q, mblk_t *emp, mblk_t *nmp);

ARGUMENTS

q

Pointer to the queue containing message emp.

emp

Enqueued message before which the new message is to be inserted
(mblk_t is an instance of the msgb(D40K) structure).

nmp

Message to be inserted.

DESCRIPTION

insq inserts a message into a queue. The message to be inserted, nmp, is placed in q immediately before the message emp. If emp is NULL, the new message is placed at the end of the queue. The queue class of the new message is ignored. All flow control parameters are updated. The service procedure is enabled unless QNOENB is set.

CAUTION: If emp is non-NULL, it must point to a message on q or a system panic could result.

RETURN VALUE

insq returns 1 on success, and 0 on failure.

LEVEL

Base or Interrupt

SEE ALSO

BCI Driver Development Guide, Chapter 7, "STREAMS"

EXAMPLE

This routine illustrates the steps a transport provider may take to place expedited data ahead of normal data on a queue (assume all M_DATA messages are converted into M_PROTO T_DATA_REQ messages). Normal T_DATA_REQ messages are just placed on the end of the queue (line 14). However, expedited T_EXDATA_REQ messages are inserted before any normal messages already on the queue (line 28). If there are no normal messages on the queue, bp will be NULL and we will fall out of the for loop (line 21). insq will act like putq(D3DK) in this case.

```c
#include <sys/tihdr.h>

xxxwput(q, mp)
    queue_t *q;
    mblk_t *mp;
{
    union T_primitives *tp;

    switch (mp->b_datap->db_type) {
    case MPROTO:
        tp = (union T_primitives *)mp->b_rptr;
        switch (tp->type) {
```
case T_DATA_REQ:
    putq(q, mp);
    break;

case T_EXDATA_REQ:
    mblk_t *bp;
    union T_primitives *ntp;
    for (bp = q->q_first; bp; bp = bp->b_next) {
        if (bp->b_datap->db_type == M_PROTO) {
            ntp = (union T_primitives *)bp->b_rptr;
            if (ntp->type != T_EXDATA_REQ)
                break;
        }
    }
    insq(q, bp, mp);
    break;

  ...
NAME

itoemajor - convert internal to external major device number

SYNOPSIS

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

int itoemajor(major_t imaj, int prevemaj);

ARGUMENTS

imaj    An internal major number.
prevemaj Most recently obtained external major number (or NODEV, if this is the first time the function has been called).

DESCRIPTION

itoemajor converts the internal major number to the external major number. The external-to-internal major number mapping is many-to-one, and so any internal major number may correspond to more than one external major number. By repeatedly invoking this function and passing the most recent external major number obtained, the driver can obtain all possible external major number values.

RETURN VALUE

External major number, or NODEV, if all have been searched

LEVEL

Base or Interrupt

SEE ALSO

getemajor(D3D), geteminor(D3D), etoimajor(D3D), getmajor(D3DK),
getminor(D3DK), makedevice(D3DK)
NAME
kmem_alloc — allocate space from kernel free memory

SYNOPSIS
#include <sys/types.h>
#include <sys/kmem.h>

_VOID *kmem_alloc(size_t size, int flag);

ARGUMENTS
size Number of bytes to allocate.
flag Determines if caller will sleep to wait for free space. Possible flags are
KM_SLEEP to sleep while waiting for free space, and KM_NOSLEEP to
return NULL if space is not available.

DESCRIPTION
The kmem_alloc function allocates a specified amount of kernel memory in bytes
and returns a pointer to the allocated memory. The flag argument determines
whether the function will sleep while waiting for free space to be released. If flag
has KM_SLEEP set, the caller may sleep until free space is available. If flag has
KM_NOSLEEP set and space is not available, NULL will be returned.

NOTE: Memory allocated by kmem_alloc is not paged. Available memory is
therefore limited. Excessive use of this memory is likely to affect overall system
performance.

RETURN VALUE
If successful, kmem_alloc returns a pointer to the allocated space. NULL is
returned if KM_NOSLEEP is set and memory cannot be allocated.

LEVEL
Base (interrupt only if KM_NOSLEEP is set in flag)

SEE ALSO
freerbuf(D3DK), getrbuf(D3DK), kmem_free(D3DK), kmem_zalloc(D3DK),
rmalloc(D3DK), rfree(D3DK), rminit(D3DK), rmsetwant(D3DK),
rmwant(D3DK)
NAME
kmem_free - free previously allocated kernel memory

SYNOPSIS
#include <sys/types.h>
#include <sys/kmem.h>

void kmem_free(_VOID *cp, size_t size);

ARGUMENTS
cp Address of the allocated storage from which to return size of allocated
memory.
size Number of bytes to free (same number of bytes as allocated by
kmem_alloc(D3DK) or kmem_zalloc(D3DK).

DESCRIPTION
This function returns size of storage to kernel free space previously allocated by
kmem_alloc(D3DK) or kmem_zalloc(D3DK). The cp and size values must specify
exactly one complete area of allocated memory. One kmem_free call must
correspond to one allocation.

RETURN VALUE
Under all conditions, no value is returned.

LEVEL
Base or Interrupt

SEE ALSO
freerbuf(D3DK), getrbuf(D3DK), kmem_alloc(D3DK), kmem_zalloc(D3DK),
rmalloc(D3DK), rmfree(D3DK), rminit(D3DK), rmsetwant(D3DK),
rmwant(D3DK)
NAME
kmem_zalloc - allocate and clear space from kernel free memory

SYNOPSIS
#include <sys/types.h>
#include <sys/kmem.h>

_VOID *kmem_zalloc(unsigned long size, unsigned long flag);

ARGUMENTS
size Number of bytes to allocate.
flag Determines if caller may sleep to wait for free space. Possible flags are
       KM_SLEEP to sleep while waiting for free space, and KM_NOSLEEP to
       return NULL if space is not available.

DESCRIPTION
This function allocates size of storage from kernel free space, clears it, and returns
a pointer to the allocated memory. If flag has KM_SLEEP set, the caller may sleep
until free space is available. If flag has KM_NOSLEEP set and space is not available,
NULL will be returned.

NOTE: Memory allocated by kmem_zalloc is not paged. Available memory is
therefore limited. Excessive use of this memory is likely to affect overall system
performance.

RETURN VALUE
kmem_zalloc returns NULL if memory cannot be allocated. Otherwise, it returns
a pointer to the allocated space.

LEVEL
Base (interrupt only if KM_NOSLEEP is set in flag)

SEE ALSO
freerbuf(D3DK), getrbuf(D3DK), kmem_alloc(D3DK), kmem_free(D3DK),
rmalloc(D3DK), rfree(D3DK), rminit(D3DK), rmsetwant(D3DK),
rmwant(D3DK)
NAME
kvtophys - convert kernel virtual address to physical address

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

paddr_t kvtophys(caddr_t caddr);

ARGUMENTS

   caddr   Kernel virtual address to translate.

DESCRIPTION
This function returns the physical address equivalent of the specified kernel vir-
tual address. The same functionality is provided in the vtop(D3D) function.

RETURN VALUE
kvtophys returns NULL if caddr is invalid; otherwise, a physical address is
returned. CAUTION: If caddr is invalid, kvtophys could panic the system.

LEVEL
Base or Interrupt

SEE ALSO
vtop(D3D)
NAME
  linkb - concatenate two message blocks

SYNOPSIS
  #include <sys/stream.h>
  void linkb(mblk_t *mp1, mblk_t *mp2);

ARGUMENTS
  mp1      The message to which mp2 is to be added. mblk_t is an instance of
           the msgb(D4DK) structure.
  mp2      The message to be added.

DESCRIPTION
  linkb creates a new message by adding mp2 to the tail of mp1. The
  continuation pointer (b_cont) of the first message is set to point to
  the second message:

  mp1
     b_datap
     b_cont
         ↓
       mp2
     b_datap
     b_cont (0)
         ↓
     db_base
     data buffer
         ↓
       db_base
       data buffer

     linkb(mp1, mp2):

RETURN VALUE
  None

LEVEL
  Base or Interrupt

SEE ALSO
  BCI Driver Development Guide, Chapter 7, "STREAMS"
  unlinkb(D3DK)

EXAMPLE
  See the dupb(D3DK) function page for an example of linkb.
NAME
makedevice - make device number from external major and minor

SYNOPSIS
#include <sys/types.h>
#include <sys/makedev.h>
#include <sys/ddi.h>

dev_t makedevice(major_t majnum, minor_t minnum);

ARGUMENTS
majnum    External major number.
minnum    External minor number.

DESCRIPTION
The makedevice function creates a device number from an external major and
external minor device number. makedevice should be used to create device
numbers so that additional overhead on the driver can be avoided, and so the
driver will port easily to releases that treat device numbers differently.

RETURN VALUE
The device number, containing both the major number and the minor number, is
returned. No validation of the external major or minor numbers is performed.
NOTE: The numbers returned by getmajor(D3DK) and getminor(D3DK) are not
valid arguments to makedevice in systems where there is a distinction between
internal and external numbers. The functions getemajor(D3D) and
genemnor(D3D) should be used on those systems.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 3, “Drivers in the UNIX Operating System”
genemfar(D3D), getemnor(D3D), getmajor(D3DK), getminor(D3DK)

EXAMPLE
In the following example makedevice creates device numbers for every device
supported by the example init(D2D) routine. The init routine initializes each
device by calling the xxx_dev_init() routine (line 8) with the device number for
each device. The device numbers are created from the preconfigured major
device number, XXMAJOR, and the range of valid minor numbers for the device.

1     xxxInit()
2     {
3         dev_t dev;
4         minor_t min;
5     
6         for (min = 0; min < XXMAXMIN; min++) {
7             dev = makedevice(XXMAJOR, min);
8             xxx_dev_init(dev);
9         }
10     }
NAME
max - return the larger of two integers

SYNOPSIS
int max(int int1, int int2);

ARGUMENTS
int1, int2 The integers to be compared.

DESCRIPTION
max compares two integers and returns the larger of two.

RETURN VALUE
The larger of the two numbers.

LEVEL
Base or Interrupt

SEE ALSO
min(D3DK)
NAME
mp_iosetup - create scatter/gather list for STREAMS drivers

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/iosystm.h>

int mp_iosetup(struct mblk *mp, struct iolist *list, int size, int maxcoalesce)

ARGUMENTS
*mp pointer to a STREAMS message structure
*list pointer to the scatter/gather list
size the number of I/O vectors in list
maxcoalesce the maximum coalescing size the controller can handle

DESCRIPTION
mp_iosetup is invoked from STREAMS driver strategy routines. It returns a count of at most size (physical address, length) I/O vectors which are returned in list, and can then be passed to a controller. The I/O vectors are computed from the virtual address in the STREAMS message. This routine ensures that the data cache has been invalidated or synced for all page(s) involved in the DMA transfer. mp_iosetup fails with a return value of -1 if list becomes larger than size.

This routine is provided specifically for device driver support on the M68000 or M88000 family of processors.

RETURN VALUE
The number of entries in list.

LEVEL
Base.

SEE ALSO
bp_iosetup(D3DK), msgb(D4DK)
NAME
   min - return the lesser of two integers

SYNOPSIS
   int min(int int1, int int2);

ARGUMENTS
   int1, int2   The integers to be compared.

DESCRIPTION
   min compares two integers and returns the lesser of the two.

RETURN VALUE
   The lesser of the two integers.

LEVEL
   Base or Interrupt

SEE ALSO
   max(D3DK)
msgdsize (D3DK)  DDI/DKI(STREAMS)  msgdsize (D3DK)

NAME
msgdsize – return the number of bytes in a message

SYNOPSIS
#include <sys/stream.h>
int msgdsize(mblk_t *mp);

ARGUMENT
mp    Message to be evaluated.

DESCRIPTION
msgdsize counts the number of bytes in a data message. Only bytes included in
the data blocks of type M_DATA are included in the count.

RETURN VALUE
The number of data bytes in a message, expressed as an integer.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”

EXAMPLE
See the bufcall(D3DK) function page for an example of the msgdsize function.
noenable(D3DK)          DDI/DKI(STREAMS)          noenable(D3DK)

NAME

noenable – prevent a queue from being scheduled

SYNOPSIS

#include <sys/stream.h>
#include <sys/ddi.h>

void noenable(queue_t *q);

ARGUMENT

q            Pointer to the queue.

DESCRIPTION

The noenable function prevents the queue q from being scheduled for service by
insq(D3DK), or by putq(D3DK) or putbq(D3DK) when enqueuing an ordinary
priority message. The queue can be re-enabled with the enableok(D3DK)
function.

RETURN VALUE

None

LEVEL

Base or Interrupt

SEE ALSO

BCI Driver Development Guide, Chapter 7, "STREAMS"

enableok(D3DK), insq(D3DK), putq(D3DK), putbq(D3DK), qenable(D3DK)
OTHERQ(D3DK)     DDI/DKI(STREAMS)     OTHERQ(D3DK)

NAME
OTHERQ - get pointer to queue's partner queue

SYNOPSIS
#include <sys/stream.h>
#include <sys/ddi.h>
queue_t *OTHERQ(queue_t *q);

ARGUMENT
q           Pointer to the queue.

DESCRIPTION
The OTHERQ function returns a pointer to the other of the two queue structures
that make up a STREAMS module or driver. If q points to the read queue the
write queue will be returned, and vice versa.

RETURN VALUE
OTHERQ returns a pointer to a queue's partner.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"

EXAMPLE
This routine sets the minimum packet size, the maximum packet size, the high
water mark, and the low water mark for the read and write queues of a given
module or driver. It is passed either one of the queues. This could be used if a
module or driver wished to update its queue parameters dynamically.

1  void
2  set_q_params(q, min, max, hi, lo)
3      queue_t *q;
4  short min;
5  short max;
6  ushort hi;
7  ushort lo;
8  {
9      q->q_minpsz = min;
10     q->q_maxpsz = max;
11     q->q_hiwat = hi;
12     q->q_lowat = lo;
13     OTHERQ(q)->q_minpsz = min;
14     OTHERQ(q)->q_maxpsz = max;
15     OTHERQ(q)->q_hiwat = hi;
16     OTHERQ(q)->q_lowat = lo;
17  }

3/91   Page 1
NAME
page_numtopp – convert page frame number to page structure

SYNOPSIS
#include <sys/types.h>
#include <vm/page.h>

page_t page_numtopp(u_int pfn);

ARGUMENT
pfn
The page frame number to be converted.

DESCRIPTION
page_numtopp converts a page frame number to its corresponding page structure.

RETURN VALUE
A pointer to the page structure is returned. If the page frame number is invalid, NULL is returned.

LEVEL
Base or Interrupt

SEE ALSO
page_pptonum(D3DK)
NAME
  page_pptonum - convert page structure to page frame number

SYNOPSIS
  #include <sys/types.h>
  #include <vm/page.h>

  u_int page_pptonum(page_t *pp);

ARGUMENT
  pp                 Pointer to a page structure.

DESCRIPTION
  page_pptonum is called to convert a page structure to its corresponding page
  frame number.

RETURN VALUE
  The page frame number corresponding to the page structure is returned. No
  error is returned. If pp (the page structure address) is invalid, the system will
  panic.

LEVEL
  Base or Interrupt

SEE ALSO,
  page_numtopp(D3DK)
physlock(D3D) (DDI) physlock(D3D)

NAME
physlock - validate and issue raw I/O request

SYNOPSIS
#include<sys/types.h>
#include <sys/buf.h>
#include <sys/errno.h>
#include <sys/uio.h>
#include <sys/cred.h>

int physlock(int (*strategy) (), struct buf *buf, dev_t dev,
             int rwflag, daddr_t nblocks, uio_t *uio_p);

ARGUMENTS
strategy Address of the driver strategy routine.
buf Pointer to the buf structure describing the I/O request. If set to NULL,
then a buffer is allocated from the buffer pool and returned to the free
list after the transfer completes.
dev Device number.
rwflag Flag indicating whether the access is a read (B_READ) or a write
(B_WRITE). Note that B_WRITE cannot be directly tested as it is 0
nblocks Number of blocks that a logical device can support, for example, a
disk partition, or tape.
uio_p Pointer to the uio structure that defines the user space of the I/O
request.

DESCRIPTION
physlock is called by the character interface to block driver read(D2DK) and
write(D2DK) routines to help perform unbuffered I/O while maintaining the
buffer header as the interface structure.
physlock performs the following functions:

verifies the requested transfer is valid by checking if the offset is at or past
the end of the device
sets up a buffer header describing the transfer
calls uiophysio() to initiate the I/O. See uiophysio(D3D).

A transfer using physlock is considered valid if the specified data location exists
on the device, and the user has specified a storage area that exists in user
memory space.

RETURN VALUE
physlock returns 0 if the result is successful, the appropriate error number upon
failure. physlock returns the ENXIO error (see Appendix A for more information)
if an attempt is made to read beyond the end of the device. If a read is per-
formed at the end of the device, 0 is returned. ENXIO is also returned if an
attempt is made to write at the end of a device or beyond the end of the device.
physlock may also return any error code produced by uiophysio.
LEVEL
Base Only  (Do not call from an interrupt routine)

SEE ALSO
dma_pageio(D3D), strategy(D2DK), uiophysio(D3D)

EXAMPLE

```c
struct dsize {
    daddr_t nblocks; /* disk partition block number */
    int cyloff;     /* starting cylinder # of partition */
} DISKsizes[16] = {

    20448, 21, /* partition 0 = cyl 21-305 */
    12888, 126, /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */

    21816, 3,  /* partition 0 = cyl 21-305 */
    21888, 1,  /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */

    20448, 21, /* partition 0 = cyl 21-305 */
    12888, 126, /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */

    20448, 21, /* partition 0 = cyl 21-305 */
    12888, 126, /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */

    20448, 21, /* partition 0 = cyl 21-305 */
    12888, 126, /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */

    20448, 21, /* partition 0 = cyl 21-305 */
    12888, 126, /* 1 = cyl 126-305 */
    9360, 175, /* 2 = cyl 175-305 */
    7200, 205, /* 3 = cyl 205-305 */
    3600, 255, /* 4 = cyl 255-305 */
    21816, 3,  /* 5 = cyl 2-305 */
    21888, 1,  /* 6 = cyl 1-305 */
    72, 1,     /* 7 = cyl 1 */
}

DISKread(dev, uio_p, cred_p) /* direct read request from block device */
{
    dev_t dev;
    uio_t  *uio_p;
    cred_t  *cred_p;
    
    register int nbiks;
    
    /* get number of blocks in the partition */
    nbiks = DISKsizes[minor(dev) & 0x7].nblocks;
    
    /*
     * Check limits of read request. If request is in
     * the limits of the disk partition, schedule direct I/O.
     */
    
    physlock(DISKstrat, 0, dev, B_READ, nbiks, uio_p);
}

} /* end DISKread */

DISKwrite(dev, uio_p, cred_p) /* direct write request to block device */
{
    dev_t dev;
    uio_t  *uio_p;
    cred_t  *cred_p
    
    register int nbiks;
    
    /* get number of blocks in the partition */
```
nblks = DISKsizes[minor(dev) & 0x7].nblocks;

/*
 * Check limits of write request. If request is in
 * the limits of the disk partition, schedule direct I/O.
 */

physiock(DISKstrat, 0, dev, B_WRITE, nblks, uio_p);

} /* end DISKwrite */
NAME
pollwakeup – inform a process that an event has occurred

SYNOPSIS
#include <sys/poll.h>
void pollwakeup(struct pollhead *php, short event);

ARGUMENTS
php Pointer to a pollhead structure.
   event Event to notify the process about.

DESCRIPTION
The pollwakeup function wakes a process waiting on the occurrence of an event.
It should be called from a driver for each occurrence of an event. The pollhead
structure will usually be associated with the driver's private data structure associ­
ated with the particular minor device where the event has occurred. See
chpoll(D2DK) and poll(2) for more detail.

RETURN
None

LEVEL
Base or Interrupt

SEE ALSO
chpoll(D2DK), poll(2)
ptob (D3DK)  DDI/DKI  ptob (D3DK)

NAME

ptob — convert size in pages to size in bytes

SYNOPSIS

#include <sys/ddi.h>

unsigned long ptob(unsigned long numpages);

ARGUMENT

numpages  Size in number of pages to convert to size in bytes.

DESCRIPTION

This function returns the number of bytes that are contained in the specified
number of pages. For example, if the page size is 2048, then ptob (2) returns
4096. ptob (0) returns 0.

RETURN VALUE

The return value is always the number of bytes in the specified number of pages.
There are no invalid input values, and no checking will be performed for
overflow in the case of a page count whose corresponding byte count cannot be
represented by an unsigned long. Rather, the higher order bits will be ignored.

LEVEL

Base or interrupt

SEE ALSO

btop(D3DK), btopr(D3DK)
NAME

pullupmsg - concatenate bytes in a message

SYNOPSIS

#include <sys/stream.h>

int pullupmsg(mblk_t *mp, int len);

ARGUMENTS

• mp Pointer to the message whose blocks are to be concatenated. mblk_t is an instance of the msgb(D4DK) structure.

• len Number of bytes to concatenate.

DESCRIPTION

pullupmsg tries to combine multiple data blocks into a single block. pullupmsg concatenates and aligns the first len data bytes of the message pointed to by mp. If len equals -1, all data is concatenated. If len bytes of the same message type cannot be found, pullupmsg fails and returns 0.

RETURN VALUE

On success, 1 is returned; on failure, 0 is returned.

LEVEL

Base or Interrupt

SEE ALSO

BCI Driver Development Guide, Chapter 7, "STREAMS"

allocb(D3DK)

EXAMPLE

This is a driver write srv(D2DK) (service) routine for a device that does not support scatter/gather DMA. For all M_DATA messages, the data will be transferred to the device with DMA.

First, try to pull up the message into one message block with the pullupmsg function (line 12). If successful, the transfer can be accomplished in one DMA job. Otherwise, it must be done one message block at a time (lines 19–22). After the data has been transferred to the device, free the message and continue processing messages on the queue.

```c
1 xxxwsrv(q)
2   queue_t *q;
3 {
4     mblk_t *mp;
5     mblk_t *tmp;
6     caddr_t dma_addr;
7     int dma_len;
8     while ((mp = getq(q)) != NULL) {
9       switch (mp->b_datap->db_type) {
10         case M_DATA:
11           if (pullupmsg(mp, -1)) {
12             dma_addr = vtop(mp->b_rptr);
13             dma_len = mp->b_wptr - mp->b_rptr;
14           }
```
15     xxx_do_dma(dma_addr, dma_len);
16     freemsg(mp);
17     break;
18     }
19     for (tmp = mp; tmp; tmp = tmp->b_cont) {
20       dma_addr = vtop(tmp->b_rptr);
21       dma_len = tmp->b_wptr - tmp->b_rptr;
22     xxx_do_dma(dma_addr, dma_len);
23     }
24     freemsg(mp);
25     break;
26     }
27     }
28     }
NAME
putbq - place a message at the head of a queue

SYNOPSIS
#include <sys/stream.h>
int putbq(queue_t *q, mblk_t *bp);

ARGUMENTS
q Pointer to the queue.
bp Pointer to the message block.

DESCRIPTION
putbq places a message at the beginning of the appropriate section of the mes­
sage queue. There are always sections for high priority and ordinary messages. If other priority bands are used, each will have its own section of the queue, in priority band order, after high priority messages and before ordinary messages. putbq can be used only for ordinary and priority band messages. High priority messages are not subject to flow control, and so cannot be put back on the queue. This function is usually called when bcanput(D3DK) or canput(D3DK) deter­mines that the message cannot be passed on to the next stream component. The flow control parameters are updated to reflect the change in the queue’s status. If QNOENB is not set, the service routine is enabled.

RETURN VALUE
putbq returns 1 on success and 0 on failure.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
bcanput(D3DK), canput(D3DK), getq(D3DK), putq(D3DK)

EXAMPLE
See the bufcall(D3DK) function page for an example of putbq.
NAME
   putctl - send a control message to a queue

SYNOPSIS
   
   \#include <sys/stream.h>
   int putctl(queue_t *q, int type);

ARGUMENTS
   q  Queue to which the message is to be sent.
   type Message type (must be control, not data type).

DESCRIPTION
   putctl tests the type argument to make sure a data type has not been specified, and
   then attempts to allocate a message block. putctl fails if type is
   M_DATA, M_DELAY, M_PROTO, or M_PCPROTO, or if a message block cannot be allo-
   cated. If successful, putctl calls the put(D2DK) routine of the queue pointed to
   by q.

RETURN VALUE
   On success, 1 is returned. If type is a data type, or if a message block cannot be
   allocated, 0 is returned.

LEVEL
   Base or Interrupt

SEE ALSO
   BCI Driver Development Guide, Chapter 7, "STREAMS"
   datamsg(D3DK), putctl1(D3DK)

EXAMPLE
   The send_ctl routine is used to pass control messages downstream. M_BREAK
   messages are handled with putctl (line 11). putctll (line 16) is used for
   M_DELAY messages, so that parm can be used to specify the length of the delay. In
   either case, if a message block cannot be allocated a variable recording the
   number of allocation failures is incremented (lines 12, 17). If an invalid message
   type is detected, cmn_err(D3DK) panics the system (line 21).

   1   void
   2   send_ctl(wrq, type, parm)
   3       queue_t *wrq;
   4       unchar type;
   5       unchar parm;
   6   {  
   7       extern int num_alloc_fail;
   8         
   9       switch (type) {
   10         case M_BREAK:
   11             if (!putctl(wrq->q_next, M_BREAK))
   12                 num_alloc_fail++;
   13             break;
   14         }
   15         case M_DELAY:
   16             if (!putctll(wrq->q_next, M_DELAY, parm))
num_alloc_fail++;  
break;

default:
    cmm_err(CE_PANIC, "send_ctl: bad message type passed");
    break;
}
NAME
putctl1 - send a control message with a one-byte parameter to a queue

SYNOPSIS
#include <sys/stream.h>

int putctl1(queue_t *q, int type, int p);

ARGUMENTS
q Queue to which the message is to be sent.
type Type of message.
p One-byte parameter.

DESCRIPTION
putctl1, like putctl(D3DK), tests the type argument to make sure a data type has not been specified, and attempts to allocate a message block. The p parameter can be used, for example, to specify how long the delay will be when an M_DELAY message is being sent. putctl1 fails if type is M_DATA, MPROTO, or M_PROTO, or if a message block cannot be allocated. If successful, putctl1 calls the put(D2DK) routine of the queue pointed to by q.

RETURN VALUE
On success, 1 is returned. 0 is returned if type is a data type, or if a message block cannot be allocated.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), datamsg(D3DK), putctl(D3DK)

EXAMPLE
See the putctl(D3DK) function page for an example of putctl1.
NAME
putnext - send a message to the next queue

SYNOPSIS
#include <sys/stream.h>
#include <sys/ddi.h>
int putnext (queue_t *q, mblk_t *mp);

ARGUMENTS
q Pointer to the queue from which the message mp will be sent.
mp Message to be passed.

DESCRIPTION
The putnext function is used to pass a message to the put(D2DK) routine of the next queue in the stream.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"

EXAMPLE
See the allocb(D3DK) function page for an example of putnext.
NAME
   putq - put a message on a queue

SYNOPSIS
   #include <sys/stream.h>
   int putq(queue_t *q, mblk_t *bp);

ARGUMENTS
   q     Pointer to the queue to which the message is to be added.
   bp    Message to be put on the queue.

DESCRIPTION
   putq is used to put messages on a driver's queue after the module's put routine
   has finished processing the message. The message is placed after any other mes­
   sages of the same priority, and flow control parameters are updated. If QNOENB is
   not set, the service routine is enabled. If no processing is done, putq can be used
   as the module's put routine.

RETURN VALUE
   putq returns 1 on success and 0 on failure.

LEVEL
   Base or Interrupt

SEE ALSO
   BCI Driver Development Guide, Chapter 7, "STREAMS"
   putbq(D3DK), qenable(D3DK), rmvq(D3DK)

EXAMPLE
   See the datamsg(D3DK) function page for an example of putq.
NAME
   qenable - enable a queue

SYNOPSIS
   #include <sys/stream.h>
   #include <sys/ddi.h>
   void qenable(queue_t *q);

ARGUMENT
   q       Pointer to the queue to be enabled.

DESCRIPTION
   qenable puts the queue pointed to by q on the linked list of those whose service
   routines are ready to be called by the STREAMS scheduler.

RETURN VALUE
   None

LEVEL
   Base or Interrupt

SEE ALSO
   BCI Driver Development Guide, Chapter 7, "STREAMS"

EXAMPLE
   See the dupb(D3DK) function page for an example of the qenable.
NAME

qreply - send a message on a stream in the reverse direction

SYNOPSIS

```c
#include <sys/stream.h>

void qreply(queue_t *q, mblk_t *bp);
```

ARGUMENTS

- `q` Pointer to the queue.
- `bp` Pointer to the message to be sent in the opposite direction.

DESCRIPTION

qreply sends a message on a stream in the opposite direction from `q`. It calls the OTHERQ function to find `q`'s module partner, and passes the message by calling the put routine of the next queue in the stream after `q`'s partner.

RETURN VALUE

None

LEVEL

Base or Interrupt

SEE ALSO

- BCI Driver Development Guide, Chapter 7, "STREAMS"
- Programmer's Guide: STREAMS
- OTHERQ, putnext

EXAMPLE

This example depicts the canonical flushing code for STREAMS drivers. The driver has a write srv (service) routine that may have messages on the queue. If it receives an M_FLUSH message (line 6), and if the FLUSHW bit is on in the first byte of the message (line 7), then the write queue is flushed (line 8) and the FLUSHW bit is turned off (line 9). If the FLUSHR bit is on, then the read queue is flushed (line 12) and the message is sent back up the read side of the stream with the qreply function (line 13). If the FLUSHR bit is off, then the message is freed (line 15). See the example for flushq for the canonical flushing code for modules.

qreply does two things. First, it calls the OTHERQ function to change pointer `q` to the module's other queue structure, reversing the direction of the flow. Then it uses that queue's `q_next` pointer to call the next module's put routine with the M_IOCNAK message.

```c
1  xxxput (q, mp)
2  queue_t *q;
3  mblk_t *mp;
4  {
5      switch(mp->b_data->db_type) {
6          case M_FLUSH:
7              if (*mp->b_rptr & FLUSHW) {
8                  flushq(q, FLUSHALL);
9                  *mp->b_rptr &= -FLUSHW;
10              }
11          }
```
if (*mp->b_rptr & FLUSHR) {
    flushq(RD(q), FLUSHALL);
    qreply(q, mp);
} else {
    freemsg(mp);
    break;
    ...
}
NAME
qsize – find the number of messages on a queue

SYNOPSIS
#include <sys/stream.h>
int qsize(queue_t *q);

ARGUMENT
q
Queue to be evaluated.

DESCRIPTION
qsize evaluates the queue q and returns the number of messages it contains.

RETURN VALUE
If there are no message on the queue, qsize returns 0. Otherwise, it returns the integer representing the number of messages on the queue.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
NAME
RD — get pointer to the read queue

SYNOPSIS
#include <sys/stream.h>
#include <sys/ddi.h>
queue_t RD(queue_t *q);

ARGUMENT
q Pointer to the write queue whose read queue is to be returned.

DESCRIPTION
The RD function accepts a write queue pointer as an argument and returns a
pointer to the read queue of the same module.
CAUTION: Make sure the argument to this function is a pointer to a write queue. RD will not check for queue type, and a system panic could result if it is not the
right type.

RETURN VALUE
The pointer to the read queue.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”

EXAMPLE
See the qreply(D3DK) function page for an example of RD.
NAME
rmalloc - allocate space from a private space management map

SYNOPSIS
#include <sys/map.h>
#include <sys/ddi.h>
#include <sys/iosystm.h>

  unsigned long rmalloc (struct map *mp, int size);

ARGUMENTS
mp       memory map from where the resource is drawn
size     number of units of the resource

DESCRIPTION
rmalloc is used by a driver to allocate space from a previously defined and ini-
tialized private space management map. The map itself is declared as a structure
using the driver prefix in the form prefixmap. Memory is initially allocated for the
map either by a data array, or by the kmem_alloc(D3DK) function. rmalloc is
one of five functions used for private map management. The other functions
include:

  rfree       return previously allocated space to a map
  rinit       define a map structure and initialize a map table
  rwant       return the number of processes waiting for free space
  rsetwant    increment the count of the number of processes waiting for
              free space in the map

The rmalloc function allocates space from a memory map in terms of arbitrary
units. The system maintains the map structure by size and index, computed in
units appropriate for the memory map. For example, units may be byte
addresses, pages of memory, or blocks. The elements of the memory map are
sorted by index, and the system uses the size member to combine adjacent objects
into one memory map entry. The system allocates objects from the memory map
on a first-fit basis. The normal return value is an unsigned long set to the value
of _m_addr from the map structure.

Memory returned by rmalloc is byte aligned only.

RETURN VALUE
Under normal conditions, rmalloc returns the base of the allocated space. Other-
wise, the rmalloc function returns a 0 if all memory map entries are already allo-
cated.

LEVEL
Base.
Interrupt if rwant is not set.
SEE ALSO

BCI Driver Development Guide, Chapter 6, "Input/Output Operations"

dma_pageio(D3D), rmfree(D3DK), rminit(D3DK), rmwant(D3DK)

EXAMPLE

The following example is a simple memory map, but it illustrates the principles of map management. A driver initializes the map table by calling both the rminit(D3DK) and rmfree(D3DK) functions. rminit(D3DK) establishes the number of slots or entries in the map, and rmfree initializes the total buffer area the map is to manage. The following example is a fragment from a hypothetical start routine and illustrates the following procedures:

Declaration of the map structure (line 4). The defined map array must be initialized to zero before calling rminit.

The use of kmem_alloc(D3DK) to allocate memory for the map. This example panics the system if the required amount of memory cannot be allocated (lines 10–14).

The use of mapinit to configure the total number of entries in the map, and of rmfree to configure the total buffer area.

```c
#define XX_MAPSIZE 12
#define XX_BUFSIZE 2560

struct map xx_map[XX_MAPSIZE]; /* Space management map for */
/* a private buffer */
...
xx_start()
/*
 * Allocate private buffer. If insufficient memory,
 * display message and halt system.
 */
{
register caddr_t bp;

if ((bp = kmem_alloc(XX_BUFSIZE, KM_NOSLEEP) == 0) |

    cmn_err(CF_PANIC, "xx_start: kmem_alloc failed before %d buffer
          allocation", XX_BUFSIZE);
/* endif */
/* Initialize space management map with number
 * of slots in map.
 */
rminit(xx_map, XX_MAPSIZE);
/*
 * Initialize space management map with total
 * buffer area it is to manage.
 */
rmfree(xx_map, XX_BUFSIZE, bp);
...```
The rmalloc(D3DK) function is then used by the driver’s read or write routine to allocate buffers for specific data transfers. If the appropriate space cannot be allocated, the rmsetwant(D3DK) function is used to wait for a free buffer and the process is put to sleep until a buffer is available. When a buffer becomes available, the rmfree(D3DK) function is called to return the buffer to the map and to wake the sleeping process (no wakeup(D3DK) call is required).

The next example illustrates the following procedures:

The size of the I/O request is calculated and stored in the size variable (lines 14–15).

While buffers are available, buffers are allocated through the rmalloc function using the size value (line 25).

If there are not enough buffers free for use, the rmsetwant(D3DK) function is called, and the process is put to sleep (lines 26–28). When a buffer becomes available, the rmfree(D3DK) function returns the buffer to the map and wakes the process.

The uiomove(D3DK) function is used to move data to the allocated buffer (line 35).

If the address passed to the uiomove function is invalid, the rmfree function is called to release the previously allocated buffer, and an EFAULT error is returned.

```c
#define XX_MAPPRIO (PZERO + 6)
#define XX_MAPSIZE 12
#define XX_BUFSIZE 2560
#define XX_MAXSIZE (XX_BUFSIZE / 4)

struct map xx_map[XX_MAPSIZE];        /* Private buffer space map */
char xx_buffer[XX_BUFSIZE];           /* driver xx_buffer area */

read(dev, uio_p, cred_p) {
    dev_t dev;
    uio_t uio_p;    /* Pointer to uio structure for I/O */
    cred_t cred_p;

    register caddr_t addr;
    register int size;

    size = min(COUNT, XX_MAXSIZE); /* Break large I/O request */
        /* into small ones */
    /*
    * Get buffer. If space is not available, then
    * request a wakeup when space is returned. Wait
    * for space; rmfree will check rmsetwant and
    * supply the wakeup call.
    */
    oldlevel = spl4();

    while((addr = (caddr_t)rmalloc(xx_map, size)) == NULL) {
```
rmalloc(D3DK)  DDI/DKI  rmalloc(D3DK)

26  rmsetwant(xx_map)
27  sleep(xx_map, XX_MAXPrio);
28 } /* endwhile */
29  splx(oldlevel);
30
31 /*
32 * Move data to buffer. If invalid address is found,
33 * return buffer to map and return error code.
34 */
35 if (uiomove(addr, size, UIO_READ, uio_p) == -1) {
36   oldlevel = spl4();
37   rmfree(xx_map, size, addr);
38   splx(oldlevel);
39   return(EFAULT);
40 } /* endif */
**NAME**

`rmfree` - release free space back into a private space management map

**SYNOPSIS**

```c
#include <sys/map.h>
#include <sys/ddi.h>

void rmfree(struct map *mp, long size, unsigned long index);
```

**ARGUMENTS**

- `*mp` pointer to the `map(D4D.K)` structure
- `size` number of units being freed
- `index` index of the first unit of the allocated resource

**DESCRIPTION**

`rmfree` releases space back into a private space management map. It is the opposite of `rmalloc(D3DK)`, which allocates space that is controlled by a private map structure.

Drivers may define private space management buffers for allocation of memory space, in terms of arbitrary units, using the `rmalloc(D3DK)`, `rmfree` and `rminit(D3DK)` functions. The drivers must include the file `map.h`. The system maintains the memory map list structure by size and index, computed in units appropriate for the memory map. For example, units may be byte addresses, pages of memory, or blocks. The elements of the memory map are sorted by index, and the system uses the size member so that adjacent objects are combined into one memory map entry. The system allocates objects from the memory map on a first-fit basis. `rmfree` frees up unallocated memory for re-use.

**RETURN VALUE**

None. However, if the `m_addr` member of the `map` structure is returned as 0, the following warning message is displayed on the console:

```
WARNING: rmfree map overflow mp lost size items at index
```

where `mp` is the hexadecimal address of the `map` structure, `size` is the decimal number of buffers freed, and `index` is the decimal address to the first buffer unit freed.

Frees a block of memory more than once may crash the system.

**LEVEL**

Base or Interrupt.

**SEE ALSO**

`rmalloc(D3DK)`, `rminit(D3DK)`, `rmwant(D3DK)`

**EXAMPLE**

See `rmalloc(D3DK).`
NAME
rminit - initialize a private space management map

SYNOPSIS
#include <sys/map.h>
#include <sys/ddi.h>

void rminit (struct map *mp, unsigned long mapsize);

ARGUMENTS
*mp Pointer to the memory map from where the resource is drawn.
mapsize Number of entries for the memory map table.

DESCRIPTION
The rminit function initializes a private map structure that can be used for the
allocation of memory space. The map itself is declared as a structure using the
driver prefix in the form prefixmap. Memory is initially allocated for the map
either by a data array, or by the kmem_alloc function.
The driver must initialize the map structure by calling rminit. However, rminit
does not cause the memory map entries to be labeled available. This must be
done through rmfree before objects can actually be allocated from the
memory map.
The system maintains the memory map list structure by size and index, computed
in units appropriate for the memory map. Units may be byte addresses, pages of
memory, or blocks. The elements of the memory map are sorted by index.
Two memory map table entries are reserved for internal system use and they are
not available for memory map use.

NOTE: The map array must be initialized to zero before calling rminit.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
rmalloc, rmwant, rmfree, rmsetwant

EXAMPLE
See rmalloc.
NAME
rmsetwant – set the map's wait flag for a wakeup

SYNOPSIS
#include <sys/map.h> #include <sys/ddi.h>
void rmsetwant(struct map *map);

ARGUMENTS
map
Pointer to the map the driver is waiting for.

DESCRIPTION
The rmsetwant function increments the counter on the wait flag of the map pointed to by map. It is typically called from the driver's read or write routine after an unsuccessful attempt to allocate space from the map using rmalloc(D3DK).

Typically, a driver will sleep on map after calling rmsetwant. When the nmfree function returns space to the map, it calls wakeup(D3DK).

RETURN VALUE
None

LEVEL
Base only

SEE ALSO
rmalloc(D3DK), nmfree(D3DK), rminit(D3DK), rmwant(D3DK), map(D4DK)

EXAMPLE
See rmalloc(D3DK).
NAME
rmvb - remove a message block from a message

SYNOPSIS
#include <sys/stream.h>
mblk_t *rmvb(mblk_t *mp, mblk_t *bp);

ARGUMENTS
*mp Message from which a block is to be removed. mblk_t is an instance of the msgb(D4DK) structure.
bp Message block to be removed.

DESCRIPTION
rmvb removes a message block (bp) from a message (mp), and returns a pointer to the altered message. The message block is not freed, merely removed from the message. It is the module or driver's responsibility to free the message block.

RETURN VALUE
If successful, a pointer to the message (minus the removed block) is returned. The pointer is NULL if bp was the only block of the message before rmvb was called. If the designated message block (bp) does not exist, -1 is returned.

LEVEL
Base or Interrupt

EXAMPLE
This routine removes all zero-length M_DATA message blocks from the given message. For each message block in the message, save the next message block (line 10). If the current message block is of type M_DATA and has no data in its buffer (line 11), then remove it from the message (line 12) and free it (line 13). In either case, continue with the next message block in the message (line 16).

```c
void xxclean(mp)
mblk_t *mp;
{
    mblk_t *tmp;
mblk_t *nmp;
    tmp = mp;
    while (tmp) {
        nmp = tmp->b_next;
        if ((tmp->b_datap->db_type == M_DATA) &&
            (tmp->b_rptr == tmp->b_wptr)) {
            rmvb(mp, tmp);
            freeb(tmp);
        }
        tmp = nmp;
    }
}
```
NAME
rmvq - remove a message from a queue

SYNOPSIS
#include <sys/stream.h>
void rmvq(queue_t *q, mblk_t *mp);

ARGUMENTS
q Queue containing the message to be removed.
mp Message to remove.

DESCRIPTION
rmvq removes a message from a queue. A message can be removed from any-
where on a queue. To prevent modules and drivers from having to deal with the
internals of message linkage on a queue, either rmvq or getq(D3DK) should be
used to remove a message from a queue.

CAUTION: Make sure that the message mp exists to avoid a possible system
panic.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"

EXAMPLE
This code fragment illustrates how one may flush one type of message from a
queue. In this case, only M_PROTO T_DATA_IND messages are flushed. For each
message on the queue, if it is an M_PROTO message (line 8) of type T_DATA_IND
(line 10), save a pointer to the next message (line 11), remove the T_DATA_IND
message (line 12) and free it (line 13). Continue with the next message in the list
(line 19).

```c
mp = q->q_first;
while (mp) {
    if (mp->b_datap->db_type == M_PROTO) {
        tp = (union T_primitives *)mp->b_rptr;
        if (tp->type == T_DATA_IND) {
            nmp = mp->b_next;
            rmvq(q, mp);
            freemsg(mp);
            mp = nmp;
        } else {
            mp = mp->b_next;
        }
    }
    else {
        mp = mp->b_next;
    }
```
18     } else {
19           mp = mp->b_next;
20     }
21   }

NAME
   rmwant - wait for free memory

SYNOPSIS
   #include <sys/map.h>
   #include <sys/ddi.h>

   unsigned long rmwant (struct map *map_p);

ARGUMENT
   map_p       Pointer to the map(D4DK) structure on which the driver is waiting for space.

DESCRIPTION
   The rmwant function returns the number of processes waiting for free space in the map.

RETURN VALUE
   The number of processes waiting for free space in the map.

LEVEL
   Base or Interrupt

SEE ALSO
   rmalloc(D3DK), rminit(D3DK), rmfree(D3DK), rmsetwant(D3DK), map(D4DK)
SAMESTR (D3DK) DDI/DKI(STREAMS) SAMESTR (D3DK)

NAME
SAMESTR - test if next queue is same type

SYNOPSIS
#include <sys/stream.h>
int SAMESTR(queue_t *q);

ARGUMENT
• q Pointer to the queue.

DESCRIPTION
The SAMESTR function is used to see if the next queue in a stream (if it exists) is the same type as the current queue (that is, both are read queues or both are write queues).

RETURN VALUE
SAMESTR returns 1 if the next queue is the same type as the current queue. It returns 0 if the next queue does not exist or if it is not the same type.

LEVEL
Base or Interrupt

SEE ALSO
OTHERQ(D3DK)
NAME

sleep – suspend process activity pending execution of an event

SYNOPSIS

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

int sleep (caddr_t event, int priority);

ARGUMENTS

event Address (signifying an event) for which the process will wait to be updated.
priority Priority that is assigned to the process when it is awakened. If priority is ORed with the defined constant PCATCH, the sleep function does not call longjmp on receipt of a signal. Instead, it returns the value 1 to the calling routine.

DESCRIPTION

sleep suspends execution of a process to await certain events such as reaching a known system state in hardware or software. For instance, when a process wants to read a device and no data is available, the driver may need to call sleep to wait for data to become available before returning. This causes the kernel to suspend executing the process that called sleep and schedule another process. The process that called sleep can be restarted by a call to the wakeup(D3DK) function with the same event specified as that used to call sleep.

A driver (with data stored in local variables) may call sleep while waiting for an event to occur. Make sure another process will not interrupt the driver and overwrite the local variables.

The event address used when calling sleep should be the address of a kernel data structure or one of the driver’s own data structures. The sleep address is an arbitrary address that has no meaning except to the corresponding wakeup function call. This does not mean that any arbitrary kernel address should be used for sleep. Doing this could conflict with other, unrelated sleep/wakeup operations in the kernel. A kernel address used for sleep should be the address of a kernel data structure directly associated with the driver I/O operation (for example, a buffer assigned to the driver).

Before a process calls sleep, the driver usually sets a flag in a driver data structure indicating the reason why sleep is being called.

The priority argument, called the sleep priority, is used for scheduling purposes when the process awakens. This parameter has critical effects on how the process that called sleep reacts to signals. If the numerical value of the sleep priority is less than or equal to the constant PZERO (defined in the sys/param.h header file), then the sleeping process will not be awakened by a signal. However, if the numerical value is greater than PZERO, the system awakens the process that called sleep prematurely (that is, before the event on which sleep was called occurred) on receipt of a non-ignored, non-held signal. In this case, it returns the value 1 to the calling routine if PCATCH is set; otherwise it does a longjmp and never returns to the driver. If the event occurred, 0 is returned.
To pick the correct sleep priority, base your decision on whether or not the process should be awakened on the receipt of a signal. If the driver calls sleep for an event that is certain to happen, the driver should use a priority numerically less than or equal to PZERO. (However, you should only use priorities less than or equal to PZERO if your driver is crucial to system operation.) If the driver calls sleep while it awaits an event that may not happen, use a priority numerically greater than PZERO.

An example of an event that may not happen is the arrival of data from a remote device. When the system tries to read data from a terminal, the terminal driver might call sleep to suspend the current process while waiting for data to arrive from the terminal. If data never arrives, the sleep call will never be answered. When a user at the terminal presses the BREAK key or hangs up, the terminal driver interrupt handler sends a signal to the reading process, which is still executing sleep. The signal causes the reading process to finish the system call without having read any data. If sleep is called with a priority value that is not awakened by signals, the process can be awakened only by a specific wakeup call. If that wakeup call never happened (the user hung up the terminal), then the process executes sleep until the system is rebooted.

Another important criteria for selecting the appropriate priority is how important the event or resource being waited for is to overall system performance. For example, disk I/O is often a bottleneck, so the priority for disk I/O is higher than most other priorities. In contrast, terminal I/O is a much lower priority. The sooner the process runs, the faster the resource will be used and freed again.

Drivers calling sleep must occasionally perform cleanup operations before returning. Typical items that need cleaning up are locked data structures that should be unlocked when the system call completes. This is done by ORing priority with PCATCH and executing sleep. If sleep returns a 1, then you can cleanup any locked structures or free any allocated resources, and return. CAUTION: If sleep is called from the driver strategy(D2DK) routine, you should OR the priority argument with PCATCH or select a priority of PZERO or less.

RETURN VALUE
If the sleep priority argument is ORed with the defined constant PCATCH, the sleep function does not call longjmp on receipt of a signal; instead, it returns the value 1 to the calling routine. If the process put in a wait state by sleep is awakened by an explicit wakeup call rather than by a signal, the sleep call returns 0.

LEVEL
Base Only  (Do not call from an interrupt routine)

SEE ALSO
BCI Driver Development Guide, Chapter 10, “Synchronizing Hardware and Software Events”
delay(D3DK), biodone(D3DK), biowait(D3DK), timeout(D3DK), untimeout(D3DK), wakeup(D3DK)
EXAMPLE

See the `timeout(D3DK)` function page for an example of `sleep`.
NAME
spl - block/allow interrupts

SYNOPSIS
#include <sys/inline.h>
int spl0();
int spl1();
int spl4();
int spl5();
int spl6();
int spl7();
int splvm();
int splhi();
int splstr();
int splitty();
int splx(int oldlevel);

ARGUMENT
oldlevel Last set priority value (only splx has an input argument).

DESCRIPTION
spl blocks or allows interrupts. When a process is executing code in a driver, the
system will not switch context from that process to another executing process
unless it is explicitly told to do so by the driver. This protects the integrity of the
kernel and driver data structures. However, the system does allow devices to
interrupt the processor and handle these interrupts immediately.

The integrity of system data structures would be destroyed if an interrupt
handler were to manipulate the same data structures as a process executing in the
driver. To prevent such problems, the kernel provides the spl functions allowing
a driver to set processor execution levels, prohibiting the handling of interrupts
below the level set.

The selection of the appropriate spl function is important. The execution level to
which the processor is set must be high enough to protect the region of code; but
this level should not be so high that it unnecessarily locks out interrupts that
need to be processed quickly. A hardware device is assigned to an interrupt
priority level depending on the type of device. By using the appropriate spl
function, a driver can inhibit interrupts from its device or other devices at the
same or lower interrupt priority levels.

The spl command changes the state of the processor status word (PSW). The
PSW stores the current processor execution level, in addition to information relating
to the operating system internals. The spl functions block out interrupts that
come in at a priority level at or below a machine-dependent interrupt priority
level. The spl functions include the following:

spl0 Restores all interrupts when executing on the base level. A driver
routine may use spl0 when the routine has been called through a
system call; that is, if it is known that the level being restored is
indeed at base level.
spl(D3D)  (DDI)  spl(D3D)

**spl**1  Used in context and process switch to protect critical code.

**spl**4  Used in character drivers to protect critical code.

**spl**5  Used in character drivers to protect critical code (this function has the same effect as **spl**4).

**spl**6  Used in block drivers to protect critical code.

**spl**7  Used in any type of driver to mask out all interrupts including the clock, and should be used very sparingly.

**splvm**  Used in memory management code to protect critical regions.

**splhi**  Used in any type of driver to mask out all interrupts including the clock, and should be used very sparingly. (This function is identical to **spl**7.)

**spltty**  Used by a TTY driver to protect critical code.

**splstr**  Used to protect STREAMS driver and module critical regions of code. This is defined to be high enough to block interrupts from the highest priority STREAMS device. **splstr** is mapped to **spltty**.

**splx**  Used to terminate a section of protected critical code. This function restores the interrupt level to the previous level specified by its argument **oldlevel**.

**NOTE:** **spl** functions should not be used in interrupt routines unless you save the old interrupt priority level in a variable as it was returned from an **spl** call. Later, **splx** must be used to restore the saved old level. Never drop the interrupt priority level below the level at which an interrupt routine was entered. For example, if an interrupt routine is entered at the interrupt priority level of an **spl**6, do not call **spl**0 through **spl**5 or the stack may become corrupted.

**RETURN VALUE**

All **spl** functions (except **splx**) return the former priority level.

**EXAMPLE**

See the `untimout(D3DK)` function page for an example of **spl**.
NAME
strlog - submit messages to the log driver

SYNOPSIS
#include <sys/stream.h>
#include <sys/strlog.h>
#include <sys/log.h>

int strlog(short mid, short sid, char level, unsigned short flags,
           char *fmt, unsigned arg1, ...);

ARGUMENTS
mid Identification number of the module or driver submitting the message.
  sid Identification number for a particular minor device.
  level Tracing level for selective screening of low priority messages.
  flags Valid flag values are:
           SL_ERROR Message is for error logger.
           SL_TRACE Message is for trace.
           SL_NOTIFY Mail copy of message to system administrator.
           SL_CONSOLE Log message to console.
           SL_FATAL Error is fatal.
           SL_WARN Error is a warning.
           SL_NOTE Error is a notice.
  fmt printf(3S) style format string. %s, %e, %g, and %G formats are not
       allowed.
  arg1 Zero or more arguments to printf.

DESCRIPTION
strlog submits formatted messages to the log(7) driver. The messages can be
retrieved with the getmsg(2) system call. The flags argument specifies the type of
the message and where it is to be sent. strace(1M) receives messages from the
log driver and sends them to the standard output. strerr(1M) receives error
messages from the log driver and appends them to a file called
/var/adm/streams/error.mm-dd, where mm-dd identifies the date of the error
message.

RETURN VALUE
strlog returns 0 if the message is not seen by all the readers, 1 otherwise.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 12, "Error Reporting"
log(7)
NAME
strqget - get information about a queue or band of the queue

SYNOPSIS
#include <sys/stream.h>

int strqget(queue_t *q, qfields_t what, unsigned char pri,
            long *valp);

ARGUMENTS
q Pointer to the queue
what Which field of the queue structure to return information about. Valid values are specified in stream.h:

typedef enum qfields {
    QHIWAT  = 0, /* q_hiwat or qb_hiwat */
    QLOWAT  = 1, /* q_lowat or qb_lowat */
    QMAXPSZ = 2, /* q_maxpsz */
    QMINPSZ = 3, /* q_minpsz */
    QCOUNT  = 4, /* q_count or qb_count */
    QFIRST  = 5, /* q_first or qb_first */
    QLAST   = 6, /* q_last or qb_last */
    QFLAG   = 7, /* q_flag or qb_flag */
    QBAD    = 8
} qfields_t;

pri Priority of request.
valp The value for the requested field.

DESCRIPTION
strqget gives drivers and modules a way to get information about a queue or a particular band of a queue without directly accessing STREAMS data structures.

RETURN VALUE
On success, 0 is returned. An error number is returned on failure.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, "STREAMS"
strqset(D3DK)
NAME
strqset - change information about a queue or band of the queue

SYNOPSIS
#include <sys/stream.h>

int strqset(queue_t *q, qfields_t what, unsigned char pri, long *val);

ARGUMENTS
q Pointer to the queue.
what Which field of the queue structure to return information about. Valid
values are specified in stream.h:

typedef enum qfields {
    QHIWAT = 0, /* q_hiwat or qb_hiwat */
    QLOWAT = 1, /* q_lowat or qb_lowat */
    QMAXPSZ = 2, /* q_maxpsz */
    QMINPSZ = 3, /* q_minpsz */
    QCOUNT = 4, /* q_count or qb_count */
    QFIRST = 5, /* q_first or qb_first */
    QLAST = 6, /* q_last or qb_last */
    QFLAG = 7, /* q_flag or qb_flag */
    QBAD = 8
} qfields_t;

pri Priority of request.
val The value for the field to be changed.

DESCRIPTION
strqset gives drivers and modules a way to change information about a queue
or a particular band of a queue without directly accessing STREAMS data struc-
tures. The fields that can be returned are defined in the enumerated type
qfields. qfields defines the following fields:

RETURN VALUE
On success, 0 is returned. An error number is returned on failure. If the what
field is read-only, EPERM is returned and the field is left unchanged.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
strqget(D3DK)
NAME
testb – check for an available buffer

SYNOPSIS
#include <sys/stream.h>
int testb(int size, int pri);

ARGUMENTS
size Size of the requested buffer.
pri Priority of the allocb request.

DESCRIPTION
testb checks to see if an allocb(D3DK) call is likely to succeed if a buffer of size
bytes at priority pri is requested. Even if testb returns successfully, the call to
allocb can fail.

RETURN VALUE
Returns 1 if a buffer of the requested size is available, and 0 if one is not.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”
allocb(D3DK), bufcall(D3DK)

EXAMPLE
In a srv(D2DK) (service) routine, if copymsg(D3DK) fails (line 6), the message is
put back on the queue (line 7) and a routine, tryagain, is scheduled to be run in
one tenth of a second (HZ/10). Then the service routine returns.

When the timeout(D3DK) function runs, if there is no message on the front of
the queue, it just returns. Otherwise, for each message block in the first message,
check to see if an allocation would succeed. If the number of message blocks
equals the number we can allocate, then enable the service procedure. Otherwise,
reschedule tryagain to run again in another tenth of a second. Note that
tryagain is merely an approximation. Its accounting may be faulty. Consider
the case of a message comprised of two 1024-byte message blocks. If there is
only one free 1024-byte message block and no free 2048-byte message blocks, then
testb will still succeed twice. If no message blocks are freed of these sizes
before the service procedure runs again, then the copymsg(D3DK) will still fail.
The reason testb is used here is because it is significantly faster than calling
copymsg. We must minimize the amount of time spent in a timeout routine.

1  xxxsrvv(q)
2   queue_t *q;
3 |
4   mblk_t *mp;
5   mblk_t *mp;
6   ...
7   if ((mp = copymsg(mp)) == NULL) {
8     putbq(q, mp);
9     timeout(tryagain, (long)q, HZ/10);
return;
...
}

tryagain(q)
{
    register int can_alloc = 0;
    register int num_blks = 0;
    register mblk_t *mp
    if (!q->q_first)
        return;
    for (mp = q->q_first; mp; mp = mp->b_cont) {
        num_blks++;
        can_alloc += testb((mp->b_datap->db_lim -
            mp->b_datap->db_base), BPRI_MED);
    }
    if (num_blks == can_alloc)
        qenable(q);
    else
        timeout(tryagain, (long) q, HZ/10);
NAME
timeout - execute a function after a specified length of time

SYNOPSIS
#include <sys/types.h>
int timeout(int (*ftn) (), caddr_t arg, long ticks);

ARGUMENTS
  ftn  Kernel function to invoke when the time increment expires.
  arg  Argument to the function.
  ticks Number of clock ticks to wait before the function is called.

DESCRIPTION
The timeout function schedules the specified function to be called after a
specified time interval. Control is immediately returned to the caller. This is use­
ful when an event is known to occur within a specific time frame, or when you
want to wait for I/O processes when an interrupt is not available or might cause
problems. For example, some robotics applications do not provide a status flag
for determining when to pump information to the robot’s controller. By using
timeout, the driver can wait a predetermined interval and then begin transfer­
ring data to the robot.

The exact time interval over which the timeout takes effect cannot be guaranteed,
but the value given is a close approximation. The function called by timeout
must adhere to the same restrictions as a driver interrupt handler. It can neither
sleep nor use previously set local variables.

RETURN VALUE
Under normal conditions, an integer timeout identifier is returned (which may, in
unusual circumstances, be set to 0). Otherwise, if the timeout table is full, the
following panic message results:

  PANIC: Timeout table overflow

The timeout function returns an identifier that may be passed to the
untimeout(D3DK) function to cancel a pending request. NOTE: No value is
returned from the called function.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 10, “Synchronizing Hardware and
Software Events”
delay(D3DK), biodone(D3DK), biowait(D3DK), sleep(D3DK),
teuntimeout(D3DK), wakeup(D3DK)

EXAMPLE
See the bufcall(D3DK) function page for an example of timeout.
**NAME**

uiomove - copy kernel data using uio(D4DK) structure

**SYNOPSIS**

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

int uiomove(caddr_t address, long nbytes, enum uio_rw rwflag,
            struct uio *uio_p);
```

**ARGUMENTS**

- `address`: Source/destination kernel address of the copy.
- `nbytes`: Number of bytes to copy.
- `rwflag`: Flag indicating read or write operation. Possible values are UIO_READ and UIO_WRITE.
- `uio_p`: Pointer to the uio structure for the copy.

**DESCRIPTION**

The `uiomove` function copies `nbytes` of data to or from the space defined by the `uio` structure (described in uio.h) and the driver.

The `uio_segflg` member of the `uio` structure determines the type of space to or from which the transfer being made. If it is set to UIO_SYSSPACE the data transfer is between addresses in the kernel. If it is set to UIO_USERSPACE the transfer is between a user program and kernel space.

In addition to moving the data, `uiomove` adds the number of bytes moved to the `iov_base` member of the `iovec` structure, decreases the `iov_len` member, increases the `uio_offset` member of the `uio` structure, and decreases the `uio_resid` member.

This function does automatic page boundary checking. `nbytes` does not have to be word-aligned.

**CAUTION:** If `uio_segflg` is set to UIO_SYSSPACE and `address` is selected from user space, the system panics.

**RETURN VALUE**

`uiomove` returns 0 upon success or -1 on failure. The driver entry point routine through which `uiomove` was called should return EFAULT if -1 is returned.

**LEVEL**

Base.

**SEE ALSO**

`uio(D4DK)`, `ureadc(D3DK)`, `uwritec(D3DK)`

**EXAMPLE**

See `rmalloc`.
NAME
uiophysio - validate and issue raw I/O request

SYNOPSIS
#include<sys/types.h>
#include <sys/buf.h>
#include <sys/errno.h>
#include <sys/uio.h>
#include <sys/cred.h>

int uiophysio(int *(strategy) (), struct buf *buf, dev_t dev, int rwflag, uio_t *uio_p);

ARGUMENTS
strategy       Address of the driver strategy routine.
buf            Pointer to the buf structure describing the I/O request.
dev            Device number.
rwflag         Flag indicating whether the access is a read (B_READ) or a write (B_WRITE). Note that B_WRITE cannot be directly tested as it is 0.
uio_p          Pointer to the uio structure that defines the user space of the I/O request.

DESCRIPTION
uiophysio is called directly by some drivers, or indirectly via physiock(D3D).
uiophysio performs the following functions:
    faults pages in and locks the pages impacted by the I/O transfer so they can not be swapped out
    calls the driver strategy(D2DK) routine passed to it
    sleeps until the transfer is complete and is awakened by the biodone(D3DK) function in the driver's interrupt routine
    performs the necessary cleanup and updates, then returns to the driver routine
A transfer using uiophysio is considered valid if the user has specified a storage area that exists in user memory space.

RETURN VALUE
uiophysio returns 0 if the result is successful, the appropriate error number upon failure. EFAULT is returned if user memory is not available. EAGAIN is returned if uiophysio could not lock pages for DMA.

LEVEL
Base Only (Do not call from an interrupt routine)

SEE ALSO
dma_pageio(D3D), strategy(D2DK), physiock(D3D)
UNLINKB (D3DK) DDI/DKI(STREAMS) UNLINKB (D3DK)

NAME
unlinkb — remove a message block from the head of a message

SYNOPSIS
#include <sys/stream.h>

mblk_t *unlinkb(mblk_t *mp);

ARGUMENT
mp Pointer to the message.

DESCRIPTION
unlinkb removes the first message block from the message pointed to by mp. A new message, minus the removed message block, is returned.

RETURN VALUE
If successful, unlinkb returns a pointer to the message with the first message block removed. If there is only one message block in the message, NULL is returned.

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 7, “STREAMS”

EXAMPLE
The routine expects to get passed an M_PROTO T_DATA_IND message. It will remove and free the M_PROTO header and return the remaining M_DATA portion of the message.

```
1  mblk_t *
2  makedata(mp)
3      mblk_t *mp;
4  {   5      mblk_t *mp;
6            mblk_t *mp;
7        return(mp);
```

Untimeout (D3DK)

NAME
untimeout – cancel previous timeout (D3DK) function call

SYNOPSIS
#include <sys/types.h>
int untimeout(int id);

ARGUMENTS
id Identification value generated by a previous timeout function call.

DESCRIPTION
untimeout cancels a pending timeout (D3DK) request.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 10, “Synchronizing Hardware and Software Events”
delay(D3DK), biodone(D3DK), biowait(D3DK), sleep(D3DK), timeout(D3DK), wakeup(D3DK)

EXAMPLE
A driver may have to repeatedly request outside help from a computer operator. The timeout function is used to delay a certain amount of time between requests. However, once the request is honored, the driver will want to cancel the timeout operation. This is done with the untimeout function.

In a driver open(D2DK) routine, after the input arguments have been verified, the status of the device is tested. If the device is not on-line, a message is displayed on the system console. The driver schedules a wakeup(D3DK) call and waits for five minutes (line 41). If the device is still not ready, the procedure is repeated.

When the device is made ready, an interrupt is generated. The driver interrupt handling routine notes there is a suspended process. It cancels the timeout request (line 59) and wakens the suspended process (line 61).

```c
struct mtu_device {
  /* layout of physical device registers */
  int control; /* physical device control word */
  int status; /* physical device status word */
  int byte_cnt; /* number of bytes to be transferred */
  paddr_t baddr; /* DMA starting physical address */
}; /* end device */
```

```c
struct mtu {
  /* magnetic tape unit logical structure */
  struct buf *mtu_head; /* pointer to I/O queue head */
  struct buf *mtu_tail; /* pointer to buffer I/O queue tail */
  int mtu_flag; /* logical status flag */
  int mtu_to_id; /* time out ID number */
};
```
...  
15 } /* end mtu */  
16  
17 extern struct mtu_device *mtu_addr[]; /* location of dev regs */  
18 extern struct mtu mtu_tbl[]; /* location of dev structs */  
19 extern int mtu_cnt;  
...  
20 mtu_open(dev, flag, type, c_ptr)  
21 dev_t dev;  
22 {  
23    register struct mtu *dp;  
24    register struct mtu_device *rp;  
25    if ((getminor(dev) >> 3) > mtu_cnt) {/* if dev doesn’t exist */  
26        return (ENXIO); /* then return error condition */  
27    } /* endif */  
28     dp = &mtu_tbl[getminor(dev)]; /* get logical device struct */  
29     if ((dp->mtu_flag & MTU_BUSY) != 0) { /* if device is in use, */  
30         return (EBUSY); /* return busy status */  
31     } /* endif */  
32  
33     dp->mtu_flag = MTU_BUSY; /* mark device in use & clear flags */  
34     rp = xx_addr[getminor(dev) >> 3]; /* get device regs */  
35     oldlevel2 = splhi();  
36     while((rp->status & MTU_LOAD) == 0) { /* while tape not loaded */  
37         /* display mount request on console */  
38         cmn_err(CE_NOTE, "!Tape MOUNT, drive %d", minor(dev) & 0x3);  
39         dp->mtu_flag = MTU_WAIT; /* indicate process suspended */  
40         dp->mtu_to_id =timeout(wakeup, dp, 5*60*HZ); /* wait 5 min */  
41         if (sleep(dp, (PCATCH | PIZERO+2)) == 1) {/*wait on tape load */  
42             /* if user aborts process, release */  
43             dp->mtu_flag = 0; /* tape device by clearing flags */  
44             untimeout(dp->mtu_to_id);  
45             splx(oldlevel2);  
46         } /* endif */  
47     } /* endwhile */  
48     /* end while */  
49     splx(oldlevel2);  
50 } /* end mtu_open */  
...  
51 mtu_int(cntr)  
52 {  
53    int cntr; /* controller that caused the interrupt */  
54    register struct mtu_device *rp = xx_addr[cntr]; /* get device regs */  
55    register struct mtu *dp = &mtu_tbl[cntr << 3 | (rp->status & 0x3)];  
...  
56     if ((dp->mtu_flag & MTU_WAIT) != 0) { /* if process is suspended */  
57         /* waiting for tape mount, */  
58         untimeout(dp->mtu_to_id); /* cancel timeout request */  
59         dp->flag &= ~MTU_WAIT; /* clear wait flag */
60       wakeup(dp);
       /* awaken suspended process */
61     } /* endif */
...
ureadc(D3DK)  DDI/DKI  ureadc(D3DK)

NAME
ureadc – add character to a uio structure

SYNOPSIS
#include <sys/uio.h>

int ureadc(int c, uio_t *uio_p);

ARGUMENTS
c The character added to the uio structure.
* uio_p Pointer to the uio(D4DK) structure.

DESCRIPTION
ureadc transfers the character c into the address space of the uio structure
pointed to by uio_p, and updates the uio structure as for uiomove(D3DK).

RETURN VALUE
0 is returned on success and EFAULT on failure.

LEVEL
Base or Interrupt

SEE ALSO
uiomove(D3DK), uwritec(D3DK), iovec(D4DK), uio(D4DK)
NAME
useracc - verify whether user has access to memory

SYNOPSIS
#include <sys/types.h>
#include <sys/buf.h>

int useracc(caddr_t base, uint count, int access);

ARGUMENTS
base The start address of the user data area
count The size of the data transfer in bytes
access A flag to determine whether the access is a read or write. The defined constant B_READ specifies a read from the device and a write to memory. This requires that the user have write access permission for the specified data area. The defined constant B_WRITE specifies a read from memory and a write to the device. It requires read access permission for the data area. (B_READ and B_WRITE are defined in the system header file sys/buf.h.)

DESCRIPTION
useracc verifies if a user has proper access to memory. It is not necessary to use useracc for buffered I/O (including use of the copyin(D3DK) and copyout(D3DK) functions).

RETURN VALUE
Under normal conditions, 1 is returned. If the user does not have the proper access permission to the memory specified, 0 is returned.

LEVEL
Base Only (Do not call from an interrupt routine)

SEE ALSO
drv_priv(D3DK)
NAME
 uwritec - remove a character from a uio structure

SYNOPSIS
 #include <sys/uio.h>
 int uwritec (uio_t *uio_p);

ARGUMENTS
 *uio_p Pointer to the uio(D4DK) structure.

DESCRIPTION
 uwritec returns a character from the uio structure pointed to by uio_p, and
 updates the uio structure as for uiomove(D3DK).

RETURN VALUE
 The next character for processing is returned on success, and -1 is returned if uio
 is empty or there is an error.

LEVEL
 Base or Interrupt

SEE ALSO
 uiomove(D3DK), ureadc(D3DK), iovec(D4DK), uio(D4DK)
NAME
   vtop - convert virtual to physical address

SYNOPSIS
   #include <sys/types.h>
   paddr_t vtop(long vaddr, proc_t *p);

ARGUMENTS
   vaddr    Virtual address to convert.
   p        Pointer to the proc(D4X) structure used by vtop to locate the informa-
            tion tables used for memory management. To indicate that the
            address is in kernel virtual space or in the virtual space of the current
            process, set p to NULL. Drivers that can transfer data directly in and
            out of user memory space must set p to the b_proc member of the
            buf(D4DK) structure.

DESCRIPTION
   vtop converts a virtual address to a physical address. When a driver receives a
   memory address from the kernel, that address is virtual. Generally, memory
   management is performed by the MMU. However, devices that access memory
directly (DMA) deal only with physical memory addresses. In such cases, the
   driver must provide the device with physical memory addresses.

   The virtual address is the memory address being translated. The vtop function
   returns the translated address.

   Similar functionality is provided by the kvtophys(D3D) function.

RETURN VALUE
   Under normal conditions, a physical address is returned. A value of -1 will be
   returned if the virtual address to be translated is not valid.

NOTE
   If the physical memory corresponding to the virtual address being translated is
   not explicitly locked (via uiophysio or physiock) or definitely known to be in the
   correct state (for example, I/O requests via the strategy entry point or known
   kernel data), the returned address will be invalid.

LEVEL
   Base or Interrupt.

SEE ALSO
   btop(D3DK), btopr(D3DK), ptob(D3DK), kvtophys(D3D)
NAME
wakeup - resume suspended process execution

SYNOPSIS
#include <sys/types.h>
void wakeup(caddr_t event);

ARGUMENT
event Address that is the same address used by sleep(D3DK) to suspend process execution.

DESCRIPTION
wakeup awakens all processes that called sleep with an address as the event argument. This lets the processes execute according to the scheduler. Ensure that the same event argument is used for both sleep and wakeup. It is recommended for code readability and for efficiency to have a one-to-one correspondence between events and sleep addresses. Also, there is usually one bit in the driver flag member that corresponds to the reason for calling sleep.

Whenever a driver calls sleep, it should test to ensure the event on which the driver called sleep occurred. There is an interval between the time the process that called sleep is awakened and the time it resumes execution where the state forcing the sleep may have been reentered. This can occur because all processes waiting for an event are awakened at the same time. The first process given control by the scheduler usually gains control of the event. All other processes awakened should recognize that they cannot continue and should reissue sleep.

RETURN VALUE
None

LEVEL
Base or Interrupt

SEE ALSO
BCI Driver Development Guide, Chapter 10, "Synchronizing Hardware and Software Events"
delay(D3DK), biodone(D3DK), biowait(D3DK), sleep(D3DK), timeout(D3DK), untimeout(D3DK)

EXAMPLE
See the untimeout(D3DK) function page for an example of wakeup.
NAME

WR — get pointer to the write queue for this module or driver

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/ddi.h>

queue_t WR(queue_t *q);
```

ARGUMENTS

- `q`: Pointer to the read queue whose write queue is to be returned.

DESCRIPTION

The WR function accepts a read queue pointer as an argument and returns a pointer to the write queue of the same module.

CAUTION: Make sure the argument to this function is a pointer to a read queue. WR will not check for queue type, and a system panic could result if the pointer is not to a read queue.

RETURN VALUE

The pointer to the write queue.

LEVEL

Base or Interrupt

SEE ALSO

Programmer's Guide: STREAMS
OTHERQ(D3DK), RD(D3DK)

EXAMPLE

In a STREAMS close routine, the driver or module is passed a pointer to the read queue. The driver must zero out the q_ptr field of both the read and write queues if it had previously initialized them in its open routine. These usually are set to the address of the module-specific data structure for the minor device.

```
1  xxxclose(q, flag)
2  queue_t *q;
3  int flag;
4  {
5    q->q_ptr = NULL;
6    WR(q)->q_ptr = NULL;
7    ...
8  }
```
4 Data Structures (D4)

Introduction 4-1

Manual Pages 4-3
## Introduction

This chapter describes the data structures used by drivers to share information between the driver and the kernel. All driver data structures shared by both DDI and DKI are identified with the (D4DK) cross reference code. All DDI-only or DKI-only structures are identified with the (D4D) or (D4K) cross reference codes respectively.

In this section, reference pages contain the following headings:

- **NAME** summarizes the structure's purpose.
- **SYNOPSIS** lists the include file that defines the structure.
- **DESCRIPTION** provides general information about the structure.
- **STRUCTURE MEMBERS** lists all accessible structure members.
- **SEE ALSO** gives sources for further information.

Table 4-1 summarizes the STREAMS structures described in this section. STREAMS structures may be used in either DDI or DKI.

### Table 4-1: STREAMS Data Structure Summary

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datab</td>
<td>STREAMS message data structure</td>
</tr>
<tr>
<td>free_rtn</td>
<td>structure specifying routine that frees non-STREAMS data buffers</td>
</tr>
<tr>
<td>module_info</td>
<td>STREAMS driver identification and limit value structure</td>
</tr>
<tr>
<td>msgb</td>
<td>STREAMS message block structure</td>
</tr>
<tr>
<td>qband</td>
<td>STREAMS queue flow control information structure</td>
</tr>
<tr>
<td>qinit</td>
<td>structure specifying STREAMS queue processing procedures</td>
</tr>
<tr>
<td>queue</td>
<td>STREAMS queue structure</td>
</tr>
<tr>
<td>streamtab</td>
<td>structure specifying qinit structures</td>
</tr>
</tbody>
</table>
Table 4-2 summarizes structures that are not specific to STREAMS I/O. These structures may be used in either DDI or DKI, except as noted.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>block I/O data transfer structure</td>
<td></td>
</tr>
<tr>
<td>cred</td>
<td>access credential structure</td>
<td></td>
</tr>
<tr>
<td>hdedata</td>
<td>hard disk error data structure</td>
<td>DDI only</td>
</tr>
<tr>
<td>iovec</td>
<td>structure specifying address and size of I/O request using uio(D4DK)</td>
<td></td>
</tr>
<tr>
<td>map</td>
<td>private memory map structure</td>
<td></td>
</tr>
<tr>
<td>uio</td>
<td>scatter/gather I/O request structure</td>
<td></td>
</tr>
</tbody>
</table>

**CAUTION** Do not declare arrays of structures as the size of the structures may change between releases. Rely only on the structure members listed in this chapter and not on unlisted members or the position of a member in a structure.
Table of Contents

4. File Formats

buf(D4DK) ..................................................................................... block I/O data transfer structure
cred(D4DK) ..................................................................................... access credential structure
datab(D4DK) ............................................................................... STREAMS message data structure
free_rtn(D4DK) .................................. structure that specifies a driver's message freeing routine
hdedata(D4D) ..................................................................................... hard disk error data structure
iovec(D4DK) ....................................................... data storage structure for I/O using uio(D4DK)
map(D4DK) ..................................................................................... private memory map structure
module_info(D4DK) ........................... STREAMS driver identification and limit value structure
msgb(D4DK) ............................................................................. STREAMS message block structure
qband(D4DK) .............................................. STREAMS queue flow control information structure
qinit(D4DK) ..................................................... STREAMS queue processing procedures structure
queue(D4DK) ........................................................................................... STREAMS queue structure
streamtab(D4DK) ................................................................ STREAMS entity declaration structure
uio(D4DK) ............................................................................... scatter/gather I/O request structure
NAME
buf - block I/O data transfer structure

SYNOPSIS
#include <sys/buf.h>

DESCRIPTION
The buf structure is the basic data structure for block I/O transfers. Each block
I/O transfer has an associated buffer header. The header contains all the buffer
control and status information. For drivers, the buffer header pointer is the sole
argument to a block driver strategy(D2DK) routine. Do not depend on the size
of the buf structure when writing a driver.

It is important to note that a buffer header may be linked in multiple lists simulta­
neously. Because of this, most of the members in the buffer header cannot be
changed by the driver, even when the buffer header is in one of the drivers’ work
lists.

Buffer headers are also used by the system for unbuffered or physical I/O for
block drivers. In this case, the buffer describes a portion of user data space that
is locked into memory (see physiock(D3D)).

Block drivers often chain block requests so that overall throughput for the device
is maximized. The av_forw and the av_back members of the buf structure can
serve as link pointers for chaining block requests.

The following figure illustrates two linked lists of buffers. The top illustration is
the bfreelist, the list of available buffers. The bottom illustration is a queue of
allocated buffers. The lined areas indicate other buffer members.

![Buffer List Diagram](image-url)
CAUTION: Buffers are a shared resource within the kernel. Drivers should read or write only the members listed in this section. Drivers that attempt to use undocumented members of the buf structure risk corrupting data in the kernel or on the device.

The paddr macro (defined in buf.h) provides access to the b_un.b_addr member of the buf structure. (b_un is a union that contains b_addr.)

The members of the buffer header available to test or set by a driver are as follows:

b_flags stores the buffer status and tells the driver whether to read or write to the device. The driver must never clear the b_flags member. If this is done, unpredictable results can occur including loss of disk sanity and the possible failure of other kernel processes.

Valid flags are as follows:

- **B_BUSY** indicates the buffer is in use.
- **B_DONE** indicates the data transfer has completed.
- **B_ERROR** indicates an I/O transfer error.
- **B_KERNBUF** indicates the buffer is allocated by the kernel and not by a driver.
- **B_PAGEIO** indicates the buffer is being used in a paged I/O request. If B_PAGEIO is set, the b_pages field of the buffer header will point to a sorted list of page structures. Also, the b_addr field of the buffer header will be offset into the first page of the page list. If B_PAGEIO is not set, the b_addr field of the buffer header will contain the kernel...
virtual address of the I/O request. The \texttt{b\_pages} field of the buffer header is not used.

\texttt{B\_PHYS} indicates the buffer header is being used for physical (direct) I/O to a user data area. The \texttt{b\_un} member contains the starting address of the user data area.

\texttt{B\_READ} indicates data is to be read from the peripheral device into main memory.

\texttt{B\_WANTED} indicates the buffer is sought for allocation.

\texttt{B\_WRITE} indicates the data is to be transferred from main memory to the peripheral device. \texttt{B\_WRITE} is a pseudo flag that occupies the same bit location as \texttt{B\_READ}. \texttt{B\_WRITE} cannot be directly tested; it is only detected as the NOT form of \texttt{B\_READ}.

\texttt{av\_forw} and \texttt{av\_back} can be used by the driver to link the buffer into driver work lists.

\texttt{b\_dev} contains the external major and minor device numbers of the device accessed. For Release 4.0, this field is replaced by the expanded device number field \texttt{b\_edev}. \texttt{b\_dev} is maintained for compatibility.

\texttt{b\_bcount} specifies the number of bytes to be transferred in both a paged and a non-paged I/O request.

\texttt{b\_addr} is either the virtual address of the I/O request, or an offset into the first page of a page list depending on whether \texttt{B\_PAGEIO} is set. If it is set, the \texttt{b\_pages} field of the buffer header will point to a sorted list of page structures and \texttt{b\_addr} will be the offset into the first page. If \texttt{B\_PAGEIO} is not set, \texttt{b\_addr} is the virtual address from which data is read or to which data is written.

\texttt{b\_blkno} identifies which logical block on the device (the device is defined by the device number) is to be accessed. The driver may have to convert this logical block number to a physical location such as a cylinder, track, and sector of a disk.

The \texttt{b\_error} with a char data type and the expanded \texttt{b\_error} with an int data type both may hold an error code that should be passed as a return code from your driver routine. \texttt{b\_error} and \texttt{b\_oerror} is set in conjunction with the \texttt{B\_ERROR} flag (set by the operating system in the \texttt{b\_flags} member). The error codes are described in Appendix A.

\texttt{b\_resid} indicates the number of bytes not transferred because of an error.

\texttt{b\_start} holds the time the I/O request was started.

\texttt{b\_proc} contains the process table entry address for the process requesting an unbuffered (direct) data transfer to a user data area (this member is set to 0 when the transfer is buffered). The process table entry is used to perform proper virtual to physical address translation of the \texttt{b\_un} member.

\texttt{b\_pages} contains a pointer to the page structure list used in a paged I/O operation.
buf(D4DK)  

DDI/DKI  

buf(D4DK)  

\texttt{b_bufsize} contains the size of the allocated buffer.  

\texttt{(*b_iiodone)} identifies a specific biodone routine to be called by the driver when the I/O is complete.  

\texttt{b_vp} identifies the vnode associated with the block.  

\textbf{SEE ALSO}  

\texttt{strategy(D2DK), physiock(D3D), brelse(D3DK), clrbuf(D3DK), iovec(D4DK), uio(D4DK)}
NAME
cred – access credential structure

SYNOPSIS
#include <sys/cred.h>

DESCRIPTION
This structure is used to check the access credentials of the process requesting
access to kernel space.

The size of the cr_groups[] array is configurable, however, its size is the same
for all cred structures. Note that cr_ngroups records the number of elements
currently in use, not the array size.

STRUCTURE MEMBERS
ushort cr_ref; /* reference count on processes using */
* cred structure. Not set by drivers. */
ushort cr_ngroups; /* number of groups in cr_groups */
uid_t cr_uid; /* effective user ID */
gid_t cr_gid; /* effective group ID */
uid_t cr_ruid; /* real user ID */
gid_t cr_rgid; /* real group ID */
uid_t cr_suid; /* "saved" user ID (from exec) */
gid_t cr_sgid; /* "saved" group ID (from exec) */
gid_t cr_groups[1]; /* supplementary groups list */

The cred structure is defined as type cred_t.

SEE ALSO
open(D2DK), close(D2DK), ioctl(D2DK), mmap(D2DK), read(D2DK),
write(D2DK), segmap(D2DK)
NAME
datab - STREAMS message data structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
The datab structure describes the data of a STREAMS message. The actual data
contained in a STREAMS message is stored in a data buffer pointed to by this
structure. A msgb (message block) structure includes a field that points to a
datab structure.

A data block can have more than one message block pointing to it at one time, so
the db_ref member keeps track of a data block's references, preventing it from
being deallocated until all message blocks are finished with it.

STRUCTURE MEMBERS
union {
    struct datab *freep; /* routine to free non-STREAMS buffer */
    struct free_rtn *frtnp;
} db_f;
unsigned char *db_base; /* first byte of buffer */
unsigned char *db_lim; /* last byte (+1) of buffer */
unsigned char db_ref; /* # of message pointers to this data */
unsigned char db_type; /* message type */
unsigned char db_iswhat; /* status of msg/data/buffer triplet */
unsigned int db_size; /* used internally */
caddr_t db_msgaddr; /* triplet msg header; points to datab */
long db_filler; /* reserved for future use */

A datab structure is defined as type dblk_t.

SEE ALSO
BCI Driver Development Guide, Chapter 4, "Header Files and Data Structures"
free_rtn(D4DK), msgb(D4DK)
NAME

free_rtn - structure that specifies a driver's message freeing routine

SYNOPSIS

#include <sys/stream.h>

DESCRIPTION

The free_rtn structure is referenced by the dp_freep member of the datab structure. When freeb(D3D) is called to free the message, the driver's message freeing routine (referenced through the free_rtn structure) is called, with arguments, to free the data buffer.

STRUCTURE MEMBERS

void (*free_func) () /* user’s freeing routine */
char *free_arg /* arguments to free_func() */

The free_rtn structure is defined as type frtn_t.

SEE ALSO

datab(D4DK), esballoa(D3DK)
NAME
hdedata - hard disk error data structure

SYNOPSIS
#include <sys/hdelog.h>

DESCRIPTION
The hdedata data structure temporarily stores hard disk error information sent to
an error queue. A hdedata structure is initialized for every disk on the system
by hdeeqd(D3D) when the system is booted. An error queue is also initialized by
hdeeqd.

When the disk driver finds an error, it provides hdelog(D3D) with the error
information. hdelog passes the hdedata structure for the error to the error
queue. This error queue is a queue of bad block reports that have not been
remapped. This queue resides in the kernel and not on the disk.

After a number or errors are accumulated, an administrator examines the list of
errors collected in the queue. If any of the errors need to be "fixed," the
administrator remaps the bad block. Remapping means that the block address is
rewritten to a defect table on the disk. Physical Description sector information
points to this defect table.

The following figure illustrates the logging of hard disk errors:

```
driver ----> hdelog ----> error queue
         \                                  /
          \                                  /
            \                                  /
            hdedata structure
```

STRUCTURE MEMBERS
- o_dev_t diskdev; /* Major/minor disk device number */
  /* (major number for character device) */
- char dskserno[12]; /* Disk pack serial number (can be all zeros) */
- daddr_t blkaddr; /* Physical block address */
  /* in machine-independent form */
- char readtype; /* Error type: CRC (cyclical redundancy check) */
  /* or ECC (error check and correction) */
- char severity; /* Severity type: marginal or unreadable */
- char badrtcnt; /* Number of unreadable tries */
- char bitwidth; /* Bitwidth of corrected error: 0 if CRC */
- time_t timestamp; /* Time stamp */

NOTE: The disk pack serial number is not currently evaluated, but it must con­
tain a value. Set to all zeros.

SEE ALSO
hdeeqd(D3D), hdelog(D3D)
NAME
iovec – data storage structure for I/O using uio(D4DK)

SYNOPSIS
#include <sys/uio.h>

DESCRIPTION
An iovec structure describes a data storage area for transfer in a uio structure. Conceptually, it may be thought of as a base address and length specification.

STRUCTURE MEMBERS
    caddr_t   iov_base; /* base address of the data storage area */
               /* represented by the iovec structure */
    int       iov_len; /* size of the data storage area in bytes */

SEE ALSO
uio(D4DK)
map(D4DK)  DDI/DKI  map(D4DK)

NAME
map - private memory map structure

SYNOPSIS
#include <sys/map.h>

DESCRIPTION
The map structure defines the size and index into a private space management map. The private map is declared as an instance of the map structure using the driver prefix in the form prefixmap. The size is defined in the m_size field as the number of arbitrary units used to make up the map. The index is defined in m_addr as the first available unit of the map.

Private maps are managed through a set five functions:

rmalloc allocates space from a defined and initialized map
rmfree returns previously allocated space to map
rminit defines a map structure and initializes a map table
rmwant returns the number of processes waiting for free space
rmsetwant increments the count of the number of processes waiting for free space in the map

Private maps can be made up of any units appropriate for the specific uses of the map. For example, units may be byte addresses, pages of memory, or blocks. The map itself does not define the resource, and the size of the map is not related to the size of the map structure.

STRUCTURE MEMBERS
unsigned long m_size /* number of units available */
unsigned long m_addr /* address of first available unit */

SEE ALSO
rmalloc(D3DK), rmfree(D3DK), rminit(D3DK), rmsetwant(D3DK), rmwant(D3DK)
NAME
module_info - STREAMS driver identification and limit value structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
When a module or driver is declared, several identification and limit values can
be set. These values are stored in the module_info structure.

The module_info structure is intended to be read-only. However, the flow con­
trol limits (mi_hiwat and mi_lowat) and the packet size limits (mi_minpsz and
mi_maxpsz) are copied to the QUEUE structure, where they may be modified.

STRUCTURE MEMBERS
ushort mi_idnum; /* module ID number */
char *mi_idname; /* module name */
short mi_minpsz; /* minimum packet size */
short mi_maxpsz; /* maximum packet size */
ushort mi_hiwat; /* high water mark */
ushort mi_lowat; /* low water mark */

The constant FMNAMESZ, limiting the length of a module's name, is currently set to
a value of eight.

SEE ALSO
queue(D4DK)
NAME
msgb - STREAMS message block structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
A STREAMS message is made up of one or more message blocks, referenced by a
pointer to a msgb structure. The b_next and b_prev pointers are used to link
messages together on a QUEUE's message queue. The b_cont pointer links
message blocks together when a message is composed of more than one block.

Each msgb structure also includes a pointer to a datab structure, the data block
(which contains pointers to the actual data of the message), and the type of the
message.

STRUCTURE MEMBERS
struct msgb  *b_next; /* next message on queue */
struct msgb  *b_prev; /* previous message on queue */
struct msgb  *b_cont; /* next message block */
unsigned char *b_rptr; /* 1st unread data byte of buffer */
unsigned char *b_wptr; /* 1st unwritten data byte of buffer */
struct datab *b_datap; /* pointer to data block */
unsigned char b_band; /* message priority */
unsigned char b_pad1; /* used internally */
unsigned short b_flag; /* used by stream head */
long b_pad2; /* used internally */

The msgb structure is defined as type mblk_t.

SEE ALSO
BCI Driver Development Guide, Chapter 4, "Header Files and Data Structures"
datab(D4DK)
NAME
qband - STREAMS queue flow control information structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
The qband structure contains flow control information for each priority band in a queue.
The qband structure is defined as type qband_t.

STRUCTURE MEMBERS
struct qband *qb_next; /* next band’s info */
ulong qb_count /* number of bytes in band */
struct msgb *qb_first; /* start of band’s data */
struct msgb *qb_last; /* end of band’s data */
ulong *qb_hiwat; /* band’s high water mark */
ulong *qb_lowat; /* band’s low water mark */
ulong *qb_flag; /* band’s status */
long *qb_padl; /* reserved for future use */

SEE ALSO
msgb(D4DK), queue(D4DK)
NAME
qinit - STREAMS queue processing procedures structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
The qinit structure contains pointers to processing procedures for a QUEUE. The 
streamtab structure for the module or driver contains pointers to one qinit 
structure for both upstream and downstream processing.

STRUCTURE MEMBERS
int (*qi_putp)(); /* put procedure */
int (*qi_srvp)(); /* service procedure */
int (*qi_qopen)(); /* open procedure */
int (*qi_qclose)(); /* close procedure */
int (*qi_qadmin)(); /* unused */
struct module_info *qi_minfo; /* module parameters */
struct module_stat *qi_mstat; /* module statistics */

SEE ALSO
BCI Driver Development Guide, Chapter 4, “Header Files and Data Structures”
queue(D4DK), streamtab(D4DK)
NAME
queue - STREAMS queue structure

SYNOPSIS
#include <sys/stream.h>

DESCRIPTION
A STREAMS driver or module consists of two queue structures, one for upstream processing (read) and one for downstream processing (write). This structure is the major building block of a stream. It contains pointers to the processing procedures, pointers to the next and previous queues in the stream, flow control parameters, and a pointer defining the position of its messages on the STREAMS scheduler list.

The queue structure is defined as type queue_t.

STRUCTURE MEMBERS
struct qinit  *q_qinfo; /* module or driver entry points */
struct msgb   *q_first; /* first message in queue */
struct msgb   *q_last;  /* last message in queue */
struct queue  *q_next;  /* next queue in stream */
struct queue  *q_link;  /* used internally */
_VOID       q_ptr;   /* pointer to private data structure */
ulong       q_count; /* approximate size of message queue */
ulong       q_flag;  /* status of queue */
long        q_minpsiz; /* smallest packet accepted by QUEUE */
long        q_maxpsiz; /* largest packet accepted by QUEUE */
ulong       q_hiwat; /* high water mark */
ulong       q_lowat; /* low water mark */
struct qband *q_bandp; /* separate flow info */
unsigned char q_nband; /* number of priority band > 0 */
unsigned char q_pad[5]; /* reserved for future use */
long        q_pad2[2]; /* reserved for future use */

SEE ALSO
msgb(D4DK), qband(D4DK)
NAME

streamtab - STREAMS entity declaration structure

SYNOPSIS

#include <sys/stream.h>

DESCRIPTION

Each STREAMS driver or module must have a streamtab structure. Drivers access this structure through the cdevsw table, and modules use the fmodsw table.

streamtab is made up of qinit structures for both the read and write queue portions of each module or driver. (Multiplexing drivers require both upper and lower qinit structures.) The qinit structure contains the entry points through which the module or driver routines are called.

Normally, the read QUEUE contains the open and close routines. Both the read and write queue can contain put and service procedures.

STRUCTURE MEMBERS

struct qinit *st_rdinit; /* read QUEUE */
struct qinit *st_wrinit; /* write QUEUE */
struct qinit *st_muxrinit; /* lower read QUEUE*/
struct qinit *st_muxwinit; /* lower write QUEUE*/

SEE ALSO

qinit(D4DK)
NAME

uio - scatter/gather I/O request structure

SYNOPSIS

#include <sys/uio.h>

DESCRIPTION

A uio structure describes an I/O request that can be broken up into different data storage areas (scatter/gather I/O). A request is a list of iovec structures (base/length pairs) indicating where in user space or kernel space the I/O data is to be read/written.

The contents of uio structures passed to the driver through the entry points should not be written by the driver. The uiomove(D3D) function takes care of all overhead related to maintaining the state of the uio structure.

STRUCTURE MEMBERS

iovec_t *uio_iov; /* pointer to the start of the iovec */
   /* list for the uio structure */
int uio_iovcnt; /* the number of iovecs in the list */
off_t uio_offset; /* offset into file where data is */
   /* transferred from or to */
short uio_segflg; /* identifies the type of I/O transfer: */
   /* UIO_SYSSPACE: kernel <-> kernel */
   /* UIO_USERSPACE: kernel <-> user */
short uio_fmode; /* file mode flags (not driver setable) */
daddr_t uio_limit; /* ulimit for file (maximum block offset). */
   /* not driver setable */
int uio_resid; /* residual count */

The uio_iov member is a pointer to the beginning of the iovec(D4DK) list for the uio. When the uio structure is passed to the driver through an entry point, the driver should not set uio_iov. When the uio structure is created by the driver, uio_iov should be initialized by the driver and not written to afterward.

SEE ALSO

iovec(D4DK)
Appendix A: Error Codes
Appendix A: Error Codes

This appendix lists the error codes that should be returned by a driver routine when an error is encountered. Table A-1 lists the error values in alphabetic order. All the error values are defined in /usr/include/sys/errno.h. In the driver open(D2D), close(D2D), ioctl(D2D), read(D2D), and write(D2D) routines, errors are passed back to the user with the return instruction at the end of the routine. In the driver strategy(D2D) routine, errors are passed back to the user by setting the b_error member of the buf(D4D) structure to the error codes.

For STREAMS ioctl routines, error numbers translate to the error numbers sent upstream in an M_IOCNAK message. For STREAMS read and write routines, error numbers translate to the error numbers sent upstream in an M_ERROR message.

NOTE: The driver print routine should not return an error code, as the function that it calls, cmn_err(D3D), is declared as void (no error is returned).
<table>
<thead>
<tr>
<th>Error Value</th>
<th>Error Description</th>
<th>Use in these Driver Routines (D2D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAGAIN</td>
<td>Kernel resources, such as the buf structure or cache memory, are not available at this time; cannot open device (device may be busy, or the system resource is not available).</td>
<td>open, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>EFAULT</td>
<td>An invalid address has been passed as an argument; memory addressing error.</td>
<td>open, close, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>EINTR</td>
<td>PCATCH set, wake with signal; sleep interrupted by signal.</td>
<td>open, close, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>EINVAL</td>
<td>An invalid argument was passed to the routine.</td>
<td>open, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>EIO</td>
<td>An invalid argument was passed to the routine.</td>
<td>open, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>ENXIO</td>
<td>An attempt was made to access a device or subdevice that does not exist (one that is not configured); an attempt was made to perform an invalid I/O operation; an incorrect minor number was specified.</td>
<td>open, close, ioctl, read, write, strategy</td>
</tr>
<tr>
<td>EPERM</td>
<td>A process attempting an operation did not have required permission.</td>
<td>open, ioctl, read, write, close</td>
</tr>
<tr>
<td>EROFS</td>
<td>An attempt was made to open for writing a read-only device.</td>
<td>open</td>
</tr>
</tbody>
</table>
Table A-2 cross references error values to the driver routines from which the error values can be returned.

<table>
<thead>
<tr>
<th>open</th>
<th>close</th>
<th>ioctl</th>
<th>read, write, and strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAGAIN</td>
<td>EFAULT</td>
<td>EAGAIN</td>
<td>EAGAIN</td>
</tr>
<tr>
<td>EFAULT</td>
<td>EINTR</td>
<td>EFAULT</td>
<td>EFAULT</td>
</tr>
<tr>
<td>EINTR</td>
<td>EIO</td>
<td>EINTR</td>
<td>EINTR</td>
</tr>
<tr>
<td>EINVAL</td>
<td>ENXIO</td>
<td>EINVAL</td>
<td>EINVAL</td>
</tr>
<tr>
<td>EIO</td>
<td>ENXIO</td>
<td>EIO</td>
<td>EIO</td>
</tr>
<tr>
<td>ENXIO</td>
<td>EPERM</td>
<td>ENXIO</td>
<td>EPERM</td>
</tr>
<tr>
<td>EPERM</td>
<td>EROFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: Migration from Release 3.2 to Release 4.0

The UNIX System V Block and Character Interface (BCI) Reference Manual defined the functions, routines, and structures appropriate for use in the UNIX System V Release 3.2 environment. Table B-1 presents all of the kernel utility functions included in the BCI followed by information about changes to the functions for Release 4.0. Most of the functions fall into one of these categories:

- No change. The function behaves the same way it did in BCI.
- Not supported. The function is not included in either DDI or DKI. No replacement is provided.
- Supported but obsolete. The function is included in DDI or DKI but a replacement is suggested.
- Macro reimplemented as function. The calling and return syntax has not changed for macros converted to functions.
- Replaced. The function is not included in either DDI or DKI but a replacement is provided.
- Renamed only. The function was renamed, but the functionality is the same as it was under the old name.
Table B-1: 3.2 to 4.0 Migration

<table>
<thead>
<tr>
<th>BCI</th>
<th>Comments</th>
<th>DDI/DKI</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjmsg</td>
<td>No change</td>
<td>adjmsg</td>
</tr>
<tr>
<td>allocb</td>
<td>For memory mapped I/O, use esballoc</td>
<td>allocb</td>
</tr>
<tr>
<td>backq</td>
<td>No change</td>
<td>backq</td>
</tr>
<tr>
<td>bcopy</td>
<td>No change</td>
<td>bcopy</td>
</tr>
<tr>
<td>brelse</td>
<td>Supported but obsolete. Allocate buffer with kmem_alloc or getzbuf(D3DK).</td>
<td>kmem_free or freerbuf</td>
</tr>
<tr>
<td>btoc</td>
<td>Replaced</td>
<td>btop, btopr</td>
</tr>
<tr>
<td>bufcall</td>
<td>Do not use with esballoc</td>
<td>bufcall</td>
</tr>
<tr>
<td>bzero</td>
<td>Word alignment no longer required</td>
<td>bzero</td>
</tr>
<tr>
<td>canon</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>canput</td>
<td>Use bcanput to test specific priority band</td>
<td>canput</td>
</tr>
<tr>
<td>clrbuf</td>
<td>buf structure has changed</td>
<td>clrbuf</td>
</tr>
<tr>
<td>cmn_err</td>
<td>No change</td>
<td>cmn_err</td>
</tr>
<tr>
<td>copyb</td>
<td>No change</td>
<td>copyb</td>
</tr>
<tr>
<td>copyin</td>
<td>Supported but obsolete. Use uiomove</td>
<td>uiomove</td>
</tr>
<tr>
<td>copymsg</td>
<td>No change</td>
<td>copymsg</td>
</tr>
<tr>
<td>copyout</td>
<td>Supported but obsolete. Use uiomove</td>
<td>uiomove</td>
</tr>
<tr>
<td>ctob</td>
<td>Replaced</td>
<td>ptob</td>
</tr>
<tr>
<td>datamsg</td>
<td>No change</td>
<td>datamsg</td>
</tr>
<tr>
<td>delay</td>
<td>No change</td>
<td>delay</td>
</tr>
<tr>
<td>dma_alloc</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>dma_breakup</td>
<td>Replaced</td>
<td>dma_pageio</td>
</tr>
<tr>
<td>drv_rfile</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>dupb</td>
<td>No change</td>
<td>dupb</td>
</tr>
<tr>
<td>dupmsg</td>
<td>No change</td>
<td>dupmsg</td>
</tr>
<tr>
<td>enableok</td>
<td>Macro reimplemented as function</td>
<td>enableok</td>
</tr>
<tr>
<td>flushq</td>
<td>Use flushband to flush specific priority band</td>
<td>flushq</td>
</tr>
<tr>
<td>freeb</td>
<td>Frees allocb and esballoc allocated</td>
<td>freeb</td>
</tr>
</tbody>
</table>
### Table B-1: 3.2 to 4.0 Migration (continued)

<table>
<thead>
<tr>
<th>BCI</th>
<th>Comments</th>
<th>DDI/DKI</th>
</tr>
</thead>
<tbody>
<tr>
<td>freemsg</td>
<td>buffers</td>
<td>freemsg</td>
</tr>
<tr>
<td>fubyte</td>
<td>Replaced</td>
<td>uiomove</td>
</tr>
<tr>
<td>fuword</td>
<td>Replaced</td>
<td>uiomove</td>
</tr>
<tr>
<td>getc</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>getcb</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>getcf</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>geteblk</td>
<td>Replaced. Use <code>kmem_alloc</code> or <code>getrbuf</code> to allocate a buffer header</td>
<td><code>kmem_alloc</code> or <code>getrbuf</code></td>
</tr>
<tr>
<td>getq</td>
<td>No change</td>
<td>getq</td>
</tr>
<tr>
<td>getvec</td>
<td>No change</td>
<td>getvec</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>No change</td>
<td>hdeeqd</td>
</tr>
<tr>
<td>hdelog</td>
<td>No change</td>
<td>hdelog</td>
</tr>
<tr>
<td>inb</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ind</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>insq</td>
<td>No change</td>
<td>insq</td>
</tr>
<tr>
<td>iodone</td>
<td>Renamed only</td>
<td>biodone</td>
</tr>
<tr>
<td>iomove</td>
<td>Replaced</td>
<td>uiomove</td>
</tr>
<tr>
<td>iowait</td>
<td>Renamed only</td>
<td>iowait</td>
</tr>
<tr>
<td>kseg</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>linkb</td>
<td>No change</td>
<td>linkb</td>
</tr>
<tr>
<td>logmsg</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>logstray</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>longjmp</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>major</td>
<td>Renamed. Macro reimplemented as function</td>
<td>getmajor</td>
</tr>
<tr>
<td>makedev</td>
<td>Renamed. Macro reimplemented as function</td>
<td>makedevice</td>
</tr>
<tr>
<td>malloc</td>
<td>Renamed only</td>
<td>rmalloc</td>
</tr>
<tr>
<td>mapinit</td>
<td>Renamed only</td>
<td>rminit</td>
</tr>
<tr>
<td>mapwant</td>
<td>Renamed only</td>
<td>rmsetwant</td>
</tr>
<tr>
<td>max</td>
<td>No change</td>
<td>max</td>
</tr>
<tr>
<td>mfree</td>
<td>Renamed only</td>
<td>rmfree</td>
</tr>
</tbody>
</table>
## Table B-1: 3.2 to 4.0 Migration (continued)

<table>
<thead>
<tr>
<th>BCI</th>
<th>Comments</th>
<th>DDI/DKI</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>No change</td>
<td>min</td>
</tr>
<tr>
<td>minor</td>
<td>Renamed. Macro reimplemented as function</td>
<td>getminor</td>
</tr>
<tr>
<td>msgdsz</td>
<td>No change</td>
<td>msgdsz</td>
</tr>
<tr>
<td>noenable</td>
<td>Macro reimplemented as function</td>
<td>noenable</td>
</tr>
<tr>
<td>OTHERQ</td>
<td>Macro reimplemented as function</td>
<td>OTHERQ</td>
</tr>
<tr>
<td>physck</td>
<td>Replaced. Functionality included in physiock</td>
<td>physio</td>
</tr>
<tr>
<td>physio</td>
<td>Replaced. Functionality included in physiock</td>
<td></td>
</tr>
<tr>
<td>psignal</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>pullupmsg</td>
<td>No change</td>
<td>pullupmsg</td>
</tr>
<tr>
<td>putbq</td>
<td>No change</td>
<td>putbq</td>
</tr>
<tr>
<td>putc</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>putcb</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>putcf</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>putctl</td>
<td>No change</td>
<td>putctl</td>
</tr>
<tr>
<td>putctll</td>
<td>No change</td>
<td>putctll</td>
</tr>
<tr>
<td>putnext</td>
<td>Macro reimplemented as function</td>
<td>putnext</td>
</tr>
<tr>
<td>putq</td>
<td>No change</td>
<td>putq</td>
</tr>
<tr>
<td>qen</td>
<td>Macro reimplemented as function</td>
<td>qen</td>
</tr>
<tr>
<td>qreply</td>
<td>No change</td>
<td>qreply</td>
</tr>
<tr>
<td>qsize</td>
<td>No change</td>
<td>qsize</td>
</tr>
<tr>
<td>RD</td>
<td>Macro reimplemented as function</td>
<td>RD</td>
</tr>
<tr>
<td>rmvb</td>
<td>No change</td>
<td>rmvb</td>
</tr>
<tr>
<td>rmvq</td>
<td>No change</td>
<td>rmvq</td>
</tr>
<tr>
<td>signal</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>sleep</td>
<td>No change</td>
<td>sleep</td>
</tr>
<tr>
<td>spl</td>
<td>No change</td>
<td>spl</td>
</tr>
<tr>
<td>splx</td>
<td>No change</td>
<td>splx</td>
</tr>
<tr>
<td>sptalloc</td>
<td>Not supported</td>
<td>kmem_alloc</td>
</tr>
<tr>
<td>sptfree</td>
<td>Not supported</td>
<td>kmem_free</td>
</tr>
</tbody>
</table>
### Table B-1: 3.2 to 4.0 Migration (continued)

<table>
<thead>
<tr>
<th>BCI</th>
<th>Comments</th>
<th>DDI/DKI</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlog</td>
<td>No change</td>
<td>strlog</td>
</tr>
<tr>
<td>subbyte</td>
<td>Replaced</td>
<td>uiomove</td>
</tr>
<tr>
<td>suser</td>
<td>Replaced</td>
<td>drv_priv</td>
</tr>
<tr>
<td>suword</td>
<td>Replaced</td>
<td>uiomove</td>
</tr>
<tr>
<td>testb</td>
<td>No change</td>
<td>testb</td>
</tr>
<tr>
<td>timeout</td>
<td>No change</td>
<td>timeout</td>
</tr>
<tr>
<td>ttclose</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttin</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttinit</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttioctl</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttopen</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttout</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttread</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttrstrrt</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>tttimeo</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttywrite</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttyflush</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttywait</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>ttspawn</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>unlinkb</td>
<td>Not supported</td>
<td>None</td>
</tr>
<tr>
<td>untimeout</td>
<td>No change</td>
<td>untimeout</td>
</tr>
<tr>
<td>useracc</td>
<td>No change</td>
<td>useracc</td>
</tr>
<tr>
<td>vtop</td>
<td>No change</td>
<td>vtop</td>
</tr>
<tr>
<td>wakeup</td>
<td>No change</td>
<td>wakeup</td>
</tr>
<tr>
<td>WR</td>
<td>Macro reimplemented as function</td>
<td>WR</td>
</tr>
</tbody>
</table>
Index

B
block I/O  3: 5-7
buffers, for raw I/O  3: 8

D
DDI/DKI (Device Driver Interface/Driver–Kernel Interface)  1: 1-5
data structures  4: 1-2
driver entry point routines  2: 1-3
error codes  A: 1-3
kernel functions  3: 1-8
migration from Release 3.2 to Release 4.0  B: 1-5
Device Driver Interface (see DDI/DKI)
DKI (Driver–Kernel Interface) (see DDI/DKI)
driver
  block  3: 5-8
  entry points  2: 1-3
  functions  3: 1-8
  porting  1: 2
  STREAMS  3: 3-4
  structures  4: 1-2
Driver–Kernel Interface (see DDI/DKI)

R
raw I/O  3: 8

S
STREAMS entry points  2: 1-3
STREAMS functions  3: 3-4
STREAMS structures  4: 1-2
<table>
<thead>
<tr>
<th>Function/Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access credential structure</td>
<td>cred(D4DK)</td>
</tr>
<tr>
<td>access to a device</td>
<td>close(D2DK)</td>
</tr>
<tr>
<td>access to a device crash dump</td>
<td>dump(D2DK)</td>
</tr>
<tr>
<td>access to a device</td>
<td>open(D2DK)</td>
</tr>
<tr>
<td>access to a device</td>
<td>start(D2D)</td>
</tr>
<tr>
<td>access to memory</td>
<td>useracc(D3DK)</td>
</tr>
<tr>
<td>activity pending execution of an event</td>
<td>sleep(D3DK)</td>
</tr>
<tr>
<td>add character to a uio structure</td>
<td>ureadc(D3DK)</td>
</tr>
<tr>
<td>address (device)</td>
<td>iomapin(D3DK)</td>
</tr>
<tr>
<td>address for a device</td>
<td>ioprobe(D3DK)</td>
</tr>
<tr>
<td>address hat_getkpfnum</td>
<td>hat_getkpfnum(D3K)</td>
</tr>
<tr>
<td>address kvtophys convert</td>
<td>kvtophys(D3D)</td>
</tr>
<tr>
<td>address locations in the kernel</td>
<td>bcopy(D3DK)</td>
</tr>
<tr>
<td>address space</td>
<td>bp_mapin(D3DK)</td>
</tr>
<tr>
<td>address space</td>
<td>bp_mapout(D3DK)</td>
</tr>
<tr>
<td>address to physical address</td>
<td>kvtophys(D3D)</td>
</tr>
<tr>
<td>allocate a message block</td>
<td>allocb(D3DK)</td>
</tr>
<tr>
<td>allocate a message block using a</td>
<td>esballoc(D3DK)</td>
</tr>
<tr>
<td>allocate and clear space from</td>
<td>kmem_zalloc(D3DK)</td>
</tr>
<tr>
<td>allocate physically contiguous</td>
<td>iomem_alloc(D3DK)</td>
</tr>
<tr>
<td>allocate space from a private space</td>
<td>rmalloc(D3DK)</td>
</tr>
<tr>
<td>allocate space from kernel free</td>
<td>kmem_alloc(D3DK)</td>
</tr>
<tr>
<td>allocate virtual address space</td>
<td>bp_mapin(D3DK)</td>
</tr>
<tr>
<td>allocated by iomem_alloc</td>
<td>iomem_free(D3DK)</td>
</tr>
<tr>
<td>allocated kernel memory</td>
<td>kmem_free(D3DK)</td>
</tr>
<tr>
<td>allocate a message block</td>
<td>allocb(D3DK)</td>
</tr>
<tr>
<td>available bucall call</td>
<td>bucall(D3DK)</td>
</tr>
<tr>
<td>available buffer</td>
<td>testb(D3DK)</td>
</tr>
<tr>
<td>available esbbcall</td>
<td>esbbcall(D3DK)</td>
</tr>
<tr>
<td>backq get pointer to the queue</td>
<td>backq(D3DK)</td>
</tr>
<tr>
<td>band bcanput test for</td>
<td>bcanput(D3DK)</td>
</tr>
<tr>
<td>band flushband flush</td>
<td>flushband(D3DK)</td>
</tr>
<tr>
<td>band of the queue strqget</td>
<td>strqget(D3DK)</td>
</tr>
<tr>
<td>band of the queue strqset</td>
<td>strqset(D3DK)</td>
</tr>
<tr>
<td>bcanput test for flow control in</td>
<td>bcanput(D3DK)</td>
</tr>
<tr>
<td>bcopy copy data between address</td>
<td>bcopy(D3DK)</td>
</tr>
<tr>
<td>becomes available bucall</td>
<td>bucall(D3DK)</td>
</tr>
<tr>
<td>behind the current queue</td>
<td>backq(D3DK)</td>
</tr>
<tr>
<td>bfselist</td>
<td>brelse(D3DK)</td>
</tr>
<tr>
<td>biodone release buffer after block</td>
<td>biodone(D3DK)</td>
</tr>
<tr>
<td>biowait suspend processes pending</td>
<td>biowait(D3DK)</td>
</tr>
<tr>
<td>block</td>
<td>allocb(D3DK)</td>
</tr>
<tr>
<td>block</td>
<td>copyb(D3DK)</td>
</tr>
<tr>
<td>block descriptor</td>
<td>dupb(D3DK)</td>
</tr>
<tr>
<td>block drivers bp_iosetup</td>
<td>bp_iosetup(D3DK)</td>
</tr>
</tbody>
</table>
block from a message ........................................ freeb(D3DK)
block from the head of a message .......................... rmvb(D3DK)
break up an I/O request into ............................... dma_sgio(D3D)
break up an I/O request for ................................. dma_sgio(D3D)
bzero clear memory for a given ............................ bzero(D3DK)
bzero from a message ......................................... adjmsg(D3DK)
bytes in a message ............................................ msgdsize(D3DK)
bytes in a message ............................................ pullupmsg(D3DK)
bytes to size in pages ....................................... btop(D3DK)
bytes to size in pages (round down) ...................... btopr(D3DK)
bytes to size in pages (round up) .......................... btopr(D3DK)
bzero clear memory for a given ............................ bzero(D3DK)
bzero from a message ......................................... adjmsg(D3DK)
bytes in a message ............................................ msgdsize(D3DK)
bytes to size in pages ....................................... btop(D3DK)
bytes to size in pages (round down) ...................... btopr(D3DK)
bytes to size in pages (round up) .......................... btopr(D3DK)

function call untimeout
queue
band of the queue strqset
ioctl control a
poll entry point for a non-STREAMS
uwritec remove a
testb
mapped device mmap
non-STREAMS character driver
bytes bzero
kmem_zalloc allocate and
execution for a specified number of
convert microseconds to
drv_hztousec convert
buffer
panic the system
driverinfo
biowait suspend processes pending
pullupmsg
linkb
display a driver message on system
clrbuf erase the
iomem_alloc allocate physically
ioctl
bcanput test for flow
qband STREAMS queue flow
putctl send a
parameter to a/
break up an I/O request for
drv_hztousec
device number etoimajor
device number itoemajor
physical address kvtophys
drv_usectohz
structure page_numtopp
frame number page_pptonum
pages (round down) btop
pages (round up) btopr
bytes ptob
vtop
copyb
copymsg
in the kernel bcopy
cancel previous timeout(D3DK) ................................... untimeout(D3DK)
canput test for room in a message .......................... canput(D3DK)
change information about a queue or ..................... strqset(D3DK)
character device .................................................... ioctl(D2DK)
character driver chpoll ........................................ chpoll(D2DK)
character from a uio structure ............................ uwritec(D3DK)
character to a uio structure .................................. ureadc(D3DK)
clock to microseconds........................................... drv_hztousec(D3DK)
clock to microseconds ......................................... drv_usectohz(D3DK)
close relinquish access to a device ...................... close(D2DK)
clrbuf erase the contents of a ......................... clrbuf(D3DK)
cmn_err display an error message or ................ cmn_err(D3DK)
communicate with device driver ....................... driverinfo(D2DK)
completion of block I/O .............................. biowait(D3DK)
concatenate bytes in a message ......................... pullupmsg(D3DK)
concatenate two message blocks ....................... linkb(D3DK)
control a character device ................................ ioctl(D2DK)
control in specified priority band ................. bcanput(D3DK)
control information structure ....................... qband(D4DK)
control message to a queue ............................. putctl(D3DK)
control message with a one-byte ...................... putctl(D3DK)
controller that does scatter/gather ................... dma_sgio(D30)
convert clock ticks to microseconds .......... drv_hztousec(D3DK)
convert external to internal major ................... etoimajor(D30)
convert internal to external major ............... itoemajor(D30)
convert kernel virtual address to ............... kvtophys(D3DK)
convert microcodes to clock ticks ............... drv_usectohz(D3DK)
convert page frame number to page ................. page_numtopp(D3DK)
convert page structure to page .................. page_pptonum(D3DK)
convert size in bytes to size in .................... btop(D3DK)
convert size in bytes to size in .................... btopr(D3DK)
convert size in pages to size in ................... ptob(D3DK)
convert virtual to physical address ............... vtop(D3DK)
copy a message block ................................. copyb(D3DK)
copy a message ........................................... copymsg(D3DK)
copy data between address locations .......... bcopy(D3DK)
program copyout copy data from a driver to a user ......................... copyout(D3DK)
copy data from a user program to a ............................... copyin(D3DK)
copy kernel data using uio(D4DK) ............................ uiomove(D3DK)
copyb copy a message block ....................................... copyb(D3DK)
copyin copy data from a user .................................. copyin(D3DK)
copymsg copy a message ......................................... copymsg(D3DK)
copyout copy data from a device to .......................... copyout(D3DK)
copyout copy data from a user to ............................. copyout(D3DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3DK)
copy data from a device ............................... read(D2DK)
copy data from a user program to a ....................... copyin(D3K

DDI/DKI Reference Manual
and minor makedevice make
getemajor get external major
getminor get external minor
get major or internal major
get minor or internal minor
convert internal to external major
open gain access to a
read read data from a
size return size of logical
start start access to a
write write data to a
map boot information into root
message on a stream in the reverse
hdedata hard
hdelog log hard
hdeeqd initialize hard
console print
the system cmn_err
into manageable units
for controller that does/
in bytes to size in pages (round
copy data from a user program to a
point for a non-STREAMS character
driverinfo communicate with device
intro introduction to
value/ module_info STREAMS
print display a
priv determine
strlog submit messages to the log
copyout copy data from a
the write queue for this module or
driver
scatter/gather list for block
free_rtn structure that specifies a
scatter/gather list for STREAMS
information
microseconds
to clock ticks
specified interval
dump routine
dump gain access to a device crash
descriptor
dup descriptor
dupmsg
dumpmsg
get major or internal major
get minor or internal minor
convert internal to external major
device number from external major makedevice(D3DK)
device number getemajor(D30)
device number getminor(D30)
device number getmajor(D3DK)
device number getminor(D3DK)
device number itoemajor(D3DK)
device number open(D2DK)
device number read(D2DK)
device number size(D2D)
device number start(D2D)
device number write(D2DK)
dev_t mapdevice(D2DK)
direction qreply send a qreply(D3DK)
disk error data structure hdedata(D4D)
disk error hdelog(D3D)
display a driver message on system print(D2DK)
display an error message or panic cmn_err(D3DK)
dma_pageio break up an I/O request dma_pageio(D3D)
dma_sg/io break up an I/O request dma_sg/io(D3D)
down) btop convert size btop(D3DK)
driver buffer copyin copyin(D3DK)
driver chpoll poll entry chpoll(D2DK)
driver entry point routines intro(D2DK)
driver identification and limit module_info(D4DK)
driver message on system console print(D2DK)
driver priveleges drv_priv(D3DK)
driver strlog(D3DK)
driver to a user program copyout(D3DK)
driver WR get pointer to WR(D3DK)
driverinfo communicate with device driverinfo(D2DK)
drivers bp_iosetup create bp_iosetup(D3DK)
driver's message freeing routine free_rtn(D4DK)
drivers mp_iosetup create mp_iosetup(D3DK)
drv_getparm retrieve kernel state drv_getparm(D3DK)
drv_hztousec convert clock ticks to drv_hztousec(D3DK)
drv_priv determine driver privilege drv_priv(D3DK)
drv_usectohz convert micros derv_usectohz(D3DK)
drv_usecwait busy-wait for drv_usecwait(D3DK)
dump gain access to a device crash dump(D2DK)
dump routine dupb(D3DK)
dupb duplicate a message block dupb(D3DK)
duplicate a message block dupb(D3DK)
duplicate a message dupmsg(D3DK)
duplicate a message dupmsg(D3DK)
qenable enable a queue qenable(D3DK)
service
streamtab STREAMS
character driver chpoll poll
intro introduction to driver
crbuf
hdedata hard disk
gterror return I/O
hdelog log hard disk
hdeeqd initialize hard disk
cmn_err display an
using a shared buffer
is available
internal major device number
pollwakeup inform a process that an
activity pending execution of an
specified length of time timeout
clock ticks delay delay process
suspend process activity pending
wakeup resume suspended process
makedevice make device number from
gtemajor get
itoimajor convert internal to
gteminor get
number etoimajor convert
queue qsize
rmsetwant set the map's wait
band bcanput test for
qband STREAMS queue
priority band flushband
specified priority band
hat_getkpfnum get page
convert page structure to page
gnumtopp convert page
free
freerbuf
message freemsg
iomem alloc iomem_free
allocate space from kernel
and clear space from kernel
rmwant wait for
memory kmem_free
space management/ rmfree release
that specifies a driver's message
a message enableok reschedule a queue for enableok(D3DK)
entity declaration structure streamtab(D4DK)
entry point for a non-STREAMS chpoll(D2DK)
entry point routines intro(D2DK)
erase the contents of a buffer crbuf(D3DK)
error data structure hdedata(D4DK)
error gterror(D3DK)
error hdelog(D3DK)
error logging hdeeqd(D3DK)
error message or panic the system cmn_err(D3DK)
esballoc allocate a message block esballoc(D3DK)
esbbcall call function when buffer esbbcall(D3DK)
etoimajor convert external to etoimajor(D3DK)
event has occurred pollwakeup(D3DK)
event sleep suspend process sleep(D3DK)
execute a function after a timeout(D3DK)
execute for a specified number of delay(D3DK)
execute an event sleep sleep(D3DK)
execution wakeup(D3DK)
external major and minor makedevice(D3DK)
external major device number getemajor(D3DK)
external major device number itoimajor(D3DK)
external minor device number geteminor(D3DK)
external to internal major device etoimajor(D3DK)
find the number of messages on a qsize(D3DK)
flag for a wakeup rmsetwant(D3DK)
flow control in specified priority bcanput(D3DK)
flow control information structure qband(D4DK)
flush messages for a specified flushband(D3DK)
flushband flush messages for a flushband(D3DK)
flushq remove messages from a queue flushq(D3DK)
frame number for kernel address hat_getkpfnum(D3K)
frame number page_pptonum page_pptonum(D3DK)
frame number to page structure page_numtopp(D3DK)
free a message block freeb(D3DK)
free a raw buffer header freerbuf(D3DK)
free all message blocks in a freemsg(D3DK)
free memory allocated by iomem_free(D3DK)
free memory kmem_alloc kmem_alloc(D3DK)
free memory kmem_zalloc allocate kmem_zalloc(D3DK)
free memory kmem_free(D3DK)
free previously allocated kernel kmem_free(D3DK)
free space back into a private rmfree(D3DK)
freeb free a message block freerbuf(D3DK)
freeing routine free_rtn structure free_rtn(D4DK)
freemsg free all message blocks in freemsg(D3DK)
freerbuf free a raw buffer header freerbuf(D3DK)
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>free_rtn</td>
<td>Structure that specifies a free_rtn function after a specified length</td>
</tr>
<tr>
<td>timeout</td>
<td>Function call for a timeout function</td>
</tr>
<tr>
<td>untimeout</td>
<td>Function call for an untimeout function</td>
</tr>
<tr>
<td>bufcall</td>
<td>Function when a buffer becomes available</td>
</tr>
<tr>
<td>esbbcall</td>
<td>Function when buffer becomes available</td>
</tr>
<tr>
<td>dump</td>
<td>Gain access to a device crash dump</td>
</tr>
<tr>
<td>open</td>
<td>Gain access to a device</td>
</tr>
<tr>
<td>major device number</td>
<td>Get major or internal device</td>
</tr>
<tr>
<td>minor device number</td>
<td>Get minor or internal device</td>
</tr>
<tr>
<td>queue</td>
<td>Get the next message from a queue</td>
</tr>
<tr>
<td>bzero</td>
<td>Clear memory for a given number of bytes</td>
</tr>
<tr>
<td>hdedata</td>
<td>Hard disk error data structure</td>
</tr>
<tr>
<td>hdelog</td>
<td>Hard disk error log</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>bzero</td>
<td>Clear memory for a given number of bytes</td>
</tr>
<tr>
<td>hdedata</td>
<td>Hard disk error data structure</td>
</tr>
<tr>
<td>hdelog</td>
<td>Hard disk error log</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>hat_getkpfnnum</td>
<td>Get page frame number</td>
</tr>
<tr>
<td>head</td>
<td>Head of a messageunlink</td>
</tr>
<tr>
<td>hdedata</td>
<td>Hard disk error data structure</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>hdedata</td>
<td>Hard disk error data structure</td>
</tr>
<tr>
<td>hdelog</td>
<td>Hard disk error log</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>hat_getkpfnnum</td>
<td>Get page frame number</td>
</tr>
<tr>
<td>inst</td>
<td>Initialize a device</td>
</tr>
<tr>
<td>strget</td>
<td>Information about a queue or band</td>
</tr>
<tr>
<td>strset</td>
<td>Information about a queue or band</td>
</tr>
<tr>
<td>drv_getparm</td>
<td>Retrieve kernel state</td>
</tr>
<tr>
<td>mapdevice</td>
<td>Map device map boot</td>
</tr>
<tr>
<td>qband</td>
<td>STREAMS queue flow control</td>
</tr>
<tr>
<td>init</td>
<td>Initialize a device</td>
</tr>
<tr>
<td>management</td>
<td>Initialize a private space</td>
</tr>
<tr>
<td>hdeeqd</td>
<td>Initialize hard disk error logging</td>
</tr>
<tr>
<td>insq</td>
<td>Insert a message into a queue</td>
</tr>
<tr>
<td>insq</td>
<td>Insert a message into a queue</td>
</tr>
<tr>
<td>init</td>
<td>Process a device interrupt</td>
</tr>
<tr>
<td>max</td>
<td>Return the larger of two integers</td>
</tr>
<tr>
<td>min</td>
<td>Return the lesser of two integers</td>
</tr>
<tr>
<td>etoimajor</td>
<td>Convert external to internal major device number</td>
</tr>
<tr>
<td>getmajor</td>
<td>Get major or internal device</td>
</tr>
<tr>
<td>getminor</td>
<td>Get minor or internal device</td>
</tr>
<tr>
<td>number</td>
<td>Get external major device</td>
</tr>
<tr>
<td>itoemajor</td>
<td>Convert internal major device to external major device</td>
</tr>
<tr>
<td>Permuted Index</td>
<td></td>
</tr>
</tbody>
</table>
int process a device
interrupt .............................................................. int(D2D)
spl block/allow
interrupts ............................................................. spl(D3D)
bio-done release buffer after block
busy-wait for specified point routines
interval drv_usecwait ................................. drv_usecwait(D3DK)
intro introduction to driver entry ......................... intro(D2DK)
routines intro
introduction to driver entry point ....................... intro(D2DK)
dcache_inval invalidate the data cache ........... dcache_inval(D3DK)
io_mapin map an I/O address (device) .......... iomapin(D3DK)
ioctl probe an I/O address for a device .......... ioprobe(D3DK)
biodone release buffer after block
I/O address for a device .......... ioprobe(D3DK)
pending completion of block
get-terror return .............................................. geterror(D3DK)
dma_sgio break up an I/O request
get-terror return................................................... geterror(D3DK)
physiock validate and issue raw
physiock(D3D)
iomapin map an I/O address (device) .......... iomapin(D3DK)
uiowait suspend processes .......... biodone(D3DK)
uiophysio validate and issue raw
strategy perform block
strategy(D2DK)
buf block
I/O data transfer structure .............. buf(D4DK)
get-terror return .............................................. geterror(D3DK)
dma_pageio break up an I/O request
I/O request into manageable units .... dma_pageio(D3D)
io_get_major convert internal to external major device number
I/O request ..................................................... I/O(D4DK)
uiowait suspend processes .......... biodone(D3DK)
io_get_major convert internal to external major device number
I/O request ..................................................... I/O(D4DK)
uiowait suspend processes .......... biodone(D3DK)
uiowait suspend processes .......... biodone(D3DK)
uiowait suspend processes .......... biodone(D3DK)
iovec data storage structure for
strategy perform block
strategy(D2DK)
ioctl control a character device .......... ioctl(D2DK)
iomapin map an I/O address (device) .......... iomapin(D3DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
ioctl control a character device .......... ioctl(D2DK)
uiophysio validate and issue raw
uiophysio(D3D)
Permuted Index

<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list for block drivers</td>
<td>bp_iotestup(D3DK)</td>
</tr>
<tr>
<td>list for STREAMS drivers</td>
<td>mp_iotestup(D3DK)</td>
</tr>
<tr>
<td>locations in the kernel</td>
<td>bcopy(D3DK)</td>
</tr>
<tr>
<td>log driver</td>
<td>strlog(D3DK)</td>
</tr>
<tr>
<td>log hard disk error</td>
<td>hdelog(D3D)</td>
</tr>
<tr>
<td>logging</td>
<td>hdeeqd(D3D)</td>
</tr>
<tr>
<td>logical device</td>
<td>size(D2D)</td>
</tr>
<tr>
<td>major and minor</td>
<td>makedevice(D3DK)</td>
</tr>
<tr>
<td>major device number</td>
<td>etoimajor(D3DK)</td>
</tr>
<tr>
<td>major device number</td>
<td>getemajor(D3DK)</td>
</tr>
<tr>
<td>major device number</td>
<td>getmajor(D3DK)</td>
</tr>
<tr>
<td>major device number</td>
<td>itoemajor(D3D)</td>
</tr>
<tr>
<td>major or internal major device</td>
<td>getmajor(D3DK)</td>
</tr>
<tr>
<td>madevicedevice make device number from</td>
<td>makedevice(D3DK)</td>
</tr>
<tr>
<td>manageable units</td>
<td>dma_pageio(D3DK)</td>
</tr>
<tr>
<td>management map</td>
<td>rmalloc(D3DK)</td>
</tr>
<tr>
<td>management map</td>
<td>rmfree(D3DK)</td>
</tr>
<tr>
<td>management map</td>
<td>rminit(D3DK)</td>
</tr>
<tr>
<td>map an I/O address (device)</td>
<td>iomapin(D3DK)</td>
</tr>
<tr>
<td>map boot information into root</td>
<td>mapdevicel(D2DK)</td>
</tr>
<tr>
<td>map device memory into user space</td>
<td>segmap(D2K)</td>
</tr>
<tr>
<td>map private memory map structure</td>
<td>map(D4DK)</td>
</tr>
<tr>
<td>map rmalloc allocate space</td>
<td>rmalloc(D3DK)</td>
</tr>
<tr>
<td>map rmfree release free space back</td>
<td>rmfree(D3DK)</td>
</tr>
<tr>
<td>map rminit initialize</td>
<td>map(D4DK)</td>
</tr>
<tr>
<td>map structure</td>
<td>map(D4DK)</td>
</tr>
<tr>
<td>mapdevice map boot information into</td>
<td>mapdevicel(D2DK)</td>
</tr>
<tr>
<td>mapped device mmap</td>
<td>mmap(D2K)</td>
</tr>
<tr>
<td>mapping for memory mapped device</td>
<td>mmap(D2K)</td>
</tr>
<tr>
<td>map's wait flag for a wakeup</td>
<td>rmsetwant(D3DK)</td>
</tr>
<tr>
<td>max return the larger of two</td>
<td>max(D3DK)</td>
</tr>
<tr>
<td>memory allocated by iomem Alloc</td>
<td>iomem_alloc(D3DK)</td>
</tr>
<tr>
<td>memory for a given number of bytes</td>
<td>bzero(D3D)</td>
</tr>
<tr>
<td>memory into user space</td>
<td>segmap(D2K)</td>
</tr>
<tr>
<td>memory iomem Alloc</td>
<td>iomem_alloc(D3DK)</td>
</tr>
<tr>
<td>memory kmem_alloc</td>
<td>kmem_alloc(D3DK)</td>
</tr>
<tr>
<td>memory kmem_free</td>
<td>kmem_free(D3DK)</td>
</tr>
<tr>
<td>memory kmem_zalloc allocate</td>
<td>kmem_zalloc(D3DK)</td>
</tr>
<tr>
<td>memory map structure</td>
<td>map(D4DK)</td>
</tr>
<tr>
<td>memory mapped device</td>
<td>mmap(D2K)</td>
</tr>
<tr>
<td>memory</td>
<td>rmwant(D3DK)</td>
</tr>
<tr>
<td>memory useracc</td>
<td>useracc(D3DK)</td>
</tr>
<tr>
<td>message</td>
<td>adjmsg(D3DK)</td>
</tr>
<tr>
<td>message at the head of a queue</td>
<td>putbq(D3DK)</td>
</tr>
<tr>
<td>message block</td>
<td>allocb(D3DK)</td>
</tr>
<tr>
<td>message block</td>
<td>copyb(D3DK)</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>dupb</td>
<td>Duplicate a message block descriptor</td>
</tr>
<tr>
<td>freeb</td>
<td>Free a message block</td>
</tr>
<tr>
<td>rmvb</td>
<td>Remove a message block from a message</td>
</tr>
<tr>
<td>message unlinkb</td>
<td>Remove a message block from the head of a message</td>
</tr>
<tr>
<td>msgb STREAMS</td>
<td>Message block structure</td>
</tr>
<tr>
<td>esballoc alloc</td>
<td>Message block using a shared buffer</td>
</tr>
<tr>
<td>freemsg free all</td>
<td>Message blocks in a message</td>
</tr>
<tr>
<td>linkb</td>
<td>Message block structure using a shared buffer</td>
</tr>
<tr>
<td>copymsg copy a</td>
<td>Message blocks</td>
</tr>
<tr>
<td>datab STREAMS</td>
<td>Message data structure</td>
</tr>
<tr>
<td>test whether a</td>
<td>Message datamsn</td>
</tr>
<tr>
<td>dupmsg duplicate a</td>
<td>Message data structure</td>
</tr>
<tr>
<td>structure</td>
<td>Message freeing routine</td>
</tr>
<tr>
<td>free all</td>
<td>Message freeing routine free_rtn</td>
</tr>
<tr>
<td>msgdsize</td>
<td>Message msgdsize</td>
</tr>
<tr>
<td>putq put a</td>
<td>Message on a queue</td>
</tr>
<tr>
<td>direction qreply</td>
<td>Message on a stream in the reverse</td>
</tr>
<tr>
<td>print display a</td>
<td>Message on system console</td>
</tr>
<tr>
<td>cmn_err display an error</td>
<td>Message or panic the system</td>
</tr>
<tr>
<td>pullupmsg</td>
<td>Message pullupmsg</td>
</tr>
<tr>
<td>concatenate bytes</td>
<td>Message pullupmsg</td>
</tr>
<tr>
<td>inq insert a</td>
<td>Message queue</td>
</tr>
<tr>
<td>datamsn test whether a</td>
<td>Return the number of bytes in a</td>
</tr>
<tr>
<td>return the number of bytes in a</td>
<td>Message queue</td>
</tr>
<tr>
<td>putct send a</td>
<td>Message queue</td>
</tr>
<tr>
<td>putct1 send a</td>
<td>Message queue</td>
</tr>
<tr>
<td>to a queue</td>
<td>Message queue</td>
</tr>
<tr>
<td>band flushband flush</td>
<td>Message queue</td>
</tr>
<tr>
<td>flushq remove</td>
<td>Messages from the preceding queue</td>
</tr>
<tr>
<td>put receive qsize find the number of srv service queued</td>
<td>Message queue</td>
</tr>
<tr>
<td>strlog submit</td>
<td>Message queue</td>
</tr>
<tr>
<td>drv_hztousec convert clock ticks to</td>
<td>Message queue</td>
</tr>
<tr>
<td>drv_usectohz convert integers</td>
<td>Message queue</td>
</tr>
<tr>
<td>getemnor get external</td>
<td>Message queue</td>
</tr>
<tr>
<td>getminor get minor or internal number from external major and getemnor</td>
<td>Message queue</td>
</tr>
<tr>
<td>number getminor get</td>
<td>Message queue</td>
</tr>
<tr>
<td>memory mapped device pointer to the write queue for this identification and limit value/</td>
<td>Message queue</td>
</tr>
<tr>
<td>list for STREAMS drivers</td>
<td>Message queue</td>
</tr>
<tr>
<td>Permuted Index</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td><strong>structure</strong> in a message</td>
<td><strong>msgb</strong> STREAMS message block</td>
</tr>
<tr>
<td><strong>getq</strong> get the</td>
<td><strong>msgsdsize</strong> return the number of bytes</td>
</tr>
<tr>
<td><strong>SAMESTR</strong> test if</td>
<td><strong>next message from a queue</strong></td>
</tr>
<tr>
<td><strong>putnext</strong> send a message to the</td>
<td><strong>next queue</strong> is same type</td>
</tr>
<tr>
<td><strong>scheduled</strong></td>
<td><strong>next queue</strong></td>
</tr>
<tr>
<td><strong>chpoll</strong> poll entry point for a</td>
<td><strong>noenable</strong> prevent a queue from being</td>
</tr>
<tr>
<td><strong>external to internal major device</strong></td>
<td><strong>non-STREAMS character driver</strong></td>
</tr>
<tr>
<td><strong>hat_getkpfnum</strong> get page frame</td>
<td><strong>number</strong> etomajor convert</td>
</tr>
<tr>
<td><strong>minor</strong> makedevice make device</td>
<td><strong>number</strong> for kernel address</td>
</tr>
<tr>
<td><strong>getemajor</strong> get external major device</td>
<td><strong>number</strong> get major or internal major device</td>
</tr>
<tr>
<td><strong>getemajor</strong> get external minor device</td>
<td><strong>number</strong> get minor or internal minor device</td>
</tr>
<tr>
<td><strong>get major or internal major device</strong></td>
<td><strong>number</strong> internal to external major device</td>
</tr>
<tr>
<td><strong>get minor or internal minor device</strong></td>
<td><strong>bzero</strong> clear memory for a given</td>
</tr>
<tr>
<td><strong>internal to external major device</strong></td>
<td><strong>msgdsize</strong> return the</td>
</tr>
<tr>
<td><strong>process execution for a specified</strong></td>
<td><strong>msgsdsize</strong> number of bytes</td>
</tr>
<tr>
<td><strong>qsize</strong> find the</td>
<td><strong>next message from a queue</strong></td>
</tr>
<tr>
<td><strong>page</strong> structure to page frame</td>
<td><strong>next queue</strong></td>
</tr>
<tr>
<td><strong>page_numtopp</strong> convert page frame</td>
<td><strong>noenable</strong> prevent a queue from being</td>
</tr>
<tr>
<td><strong>inform</strong> a process that an event has</td>
<td><strong>non-STREAMS character driver</strong></td>
</tr>
<tr>
<td><strong>/send</strong> a control message with a</td>
<td></td>
</tr>
<tr>
<td><strong>partner queue</strong></td>
<td></td>
</tr>
<tr>
<td><strong>address</strong> hat_getkpfnum get</td>
<td></td>
</tr>
<tr>
<td>convert page structure to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>page_numtopp</strong> convert</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>convert</strong> page frame number to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>page_numtopp</strong> convert</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>number</strong> to page structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>to page frame number</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>convert</strong> size in bytes to size in</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ptob</strong> convert size in</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cmn_err</strong> display an error message or a control message with a one-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OTHERQ</strong> get pointer to queue's biowait suspend processes</td>
<td></td>
</tr>
<tr>
<td><strong>sleep</strong> suspend process activity</td>
<td></td>
</tr>
<tr>
<td><strong>strategy</strong></td>
<td></td>
</tr>
<tr>
<td><strong>convert</strong> kernel virtual address to</td>
<td></td>
</tr>
</tbody>
</table>
vttop convert virtual to physical address ......................................................... vttop(D3D)
physically contiguous memory ............................................ iomem_alloc(D3DK)
physiock validate and issue raw I/O ................................. physiock(D3D)
place a message at the head of a queue .............................. putbq(D3DK)
point for a non-STREAMS character ................................. chpoll(D2DK)
point routines .............................................................. intro(D2DK)
pointer to queue's partner queue ................................. OTHERQ(D3DK)
pointer to the queue behind the scheduled one ............................... backq(D3DK)
pointer to the read queue ................................................................. RD(D3DK)
pointer to the write queue for this event .............................. WR(D3DK)
poll entry point for a non-STREAMS character .... ................... chpoll(D3DK)
pollwakeup inform a process that an event has occurred ....... pollwakeup(D3DK)
preceding queue .............................................................. put(D2DK)
prevent a queue from being enable ....................................... noenable(D3DK)
previous timeout(D3DK) function ........................................... untimeout(D3DK)
previously allocated kernel memory ...................................... kmem_free(D3DK)
print display a driver message on.................................. print(D2DK)
priority band bcanput ............................................ bcanput(D3DK)
priority band flushband ..................................... flushband(D3DK)
private memory map structure ..................................... map(D4DK)
private space management map .............................................. rmalloc(D3DK)
private space management map ............................................. rmfree(D3DK)
private space management map ............................................ rminit(D3DK)
privileged driver ............................................................... drv_priv(D3DK)
probe an I/O address for a device ....................... ioprobe(D3DK)
procedures structure ...................................................... qinit(D4DK)
process a device interrupt .............................................. int(D2D)
process activity pending execution ......................... sleep(D3DK)
process execution for a specified interrupt ............................................. suspend(D3DK)
process execution ..................................................... wakeup(D3DK)
process that an event has occurred .................. pollwakeup(D3DK)
processes biodone release ...................... biodone(D3DK)
processes pending completion of ......................... biowait(D3DK)
processing procedures structure ..................... qinit(D4DK)
program copyout .................................................... copyout(D3DK)
program to a driver buffer ....................................... copyin(D3DK)
ptob convert size in pages to size ....................... ptob(D3DK)
pullupmsg concatenate bytes in a message ................. pullupmsg(D3DK)
put a message on a queue ............................................ putq(D3DK)
put receive messages from the preceding queue of a queue .......... put(D2DK)
putbq place a message at the head of a queue ................. putbq(D3DK)
putctl send a control message to a control buffer ................. putctl(D3DK)
putto_send a control message with a one-byte parameter to a queue .......... putto_send(D3DK)
putnext send a message to the next queue .......... putnext(D3DK)
putq put a message on a queue .................................. putq(D3DK)
qbnd STREAMS queue flow control ...................... qband(D4DK)
qenable enable a queue ........................................... qenable(D3DK)
procedures structure
in the reverse direction
on a queue
to the queue behind the current
backq get pointer to the
canput test for room in a message
structure qband STREAMS
flushq remove messages from a
enableok reschedule a
WR get pointer to the write
noenable prevent a
getq get the next message from a
SAMESTR test if next
strqget get information about a
strqset change information about a
get pointer to queue’s partner
structure qinit STREAMS
place a message at the head of a
putctl send a control message to a
with a one-byte parameter to a
receive messages from the preceding
putnext send a message to the next
putq put a message on a
qenable enable a
find the number of messages on a
RD get pointer to the read
rmvq remove a message from a
about a queue or band of the
about a queue or band of the
queue STREAMS
srv service
OTHERQ get pointer to
freerbuf free a
getbuf get a
physiock validate and issue
uiophysio validate and issue
read
RD get pointer to the
queue put
wakeup processes biodone
private space management/
rmfree
close
structure uwritec

qinit STREAMS queue processing .......... qinit(D4DK)
qreply send a message on a stream .......... qreply(D3DK)
qsize find the number of messages .......... qsize(D3DK)
queue backq get pointer .......... backq(D3DK)
queue behind the current queue .......... backq(D3DK)
queue .......... canput(D3DK)
queue flow control information .......... qband(D4DK)
queue .......... flushq(D3DK)
queue for service .......... enableok(D3DK)
queue for this module or driver .......... WR(D3DK)
queue from being scheduled .......... noenable(D3DK)
queue .......... getq(D3DK)
queue .......... insq(D3DK)
queue is same type .......... SAMESTR(D3DK)
queue or band of the queue .......... strqget(D3DK)
queue or band of the queue .......... strqset(D3DK)
queue OTHERQ .......... OTHERQ(D3DK)
queue processing procedures .......... qinit(D4DK)
queue putbq .......... putbq(D3DK)
queue putctl .......... putctl(D3DK)
queue put .......... put(D2DK)
queue .......... putnext(D3DK)
queue .......... putq(D3DK)
queue .......... qenable(D3DK)
queue qsize .......... qsize(D3DK)
queue .......... RD(D3DK)
queue .......... rmvq(D3DK)
queue STREAMS queue structure .......... queue(D4DK)
queue strqget get information .......... strqget(D3DK)
queue strqset change information .......... strqset(D3DK)
queue structure .......... queue(D4DK)
queued messages .......... srv(D2DK)
queue’s partner queue .......... OTHERQ(D3DK)
raw buffer header .......... freerbuf(D3DK)
raw buffer header .......... getbuf(D3DK)
raw I/O request .......... physiock(D3D)
raw I/O request .......... uiophysio(D3D)
RD get pointer to the read queue .......... RD(D3DK)
read data from a device .......... read(D2DK)
read queue .......... RD(D3DK)
read data from a device .......... read(D2DK)
receive messages from the preceding .......... put(D2DK)
release buffer after block I/O and .......... biodone(D3DK)
release free space back into a .......... rmfree(D3DK)
relinquish access to a device .......... close(D2DK)
remove a character from a uio .......... uwritec(D3DK)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message rmvb</td>
<td>remove a message block from a</td>
</tr>
<tr>
<td>head of a message unlinkb</td>
<td>remove a message block from the</td>
</tr>
<tr>
<td>rmvq</td>
<td>remove a message from a queue</td>
</tr>
<tr>
<td>flushq</td>
<td>remove messages from a queue</td>
</tr>
<tr>
<td>dma_sgio break up an I/O</td>
<td>request for controller that does/</td>
</tr>
<tr>
<td>dma_pageio break up an I/O</td>
<td>request into manageable units</td>
</tr>
<tr>
<td>physiock validate and issue raw I/O</td>
<td>request</td>
</tr>
<tr>
<td>uio scatter/gather I/O</td>
<td>request structure</td>
</tr>
<tr>
<td>validate and issue raw I/O</td>
<td>request uiophysio</td>
</tr>
<tr>
<td>enableok</td>
<td>resume suspended process execution</td>
</tr>
<tr>
<td>wakeup</td>
<td>return buffer to the bfreelist</td>
</tr>
<tr>
<td>drv_getparm</td>
<td>return 1/0 error</td>
</tr>
<tr>
<td>brelse</td>
<td>return size of logical device</td>
</tr>
<tr>
<td>geterror</td>
<td>return the larger of two integers</td>
</tr>
<tr>
<td>size</td>
<td>return the lesser of two integers</td>
</tr>
<tr>
<td>min</td>
<td>return the number of bytes in a</td>
</tr>
<tr>
<td>message msgdsize</td>
<td>reverse direction qreply</td>
</tr>
<tr>
<td>send a message on a stream in the private space management map</td>
<td>malloc allocate space from a</td>
</tr>
<tr>
<td>a private space management map</td>
<td>rmfree release free space back into</td>
</tr>
<tr>
<td>management map</td>
<td>rtninit initialize a private space</td>
</tr>
<tr>
<td>for a wakeup message</td>
<td>rmwant set the map's wait flag</td>
</tr>
<tr>
<td>canput test for</td>
<td>rmvq remove a message block from a</td>
</tr>
<tr>
<td>mapdevice map boot information into size in bytes to size in pages</td>
<td>room in a message queue</td>
</tr>
<tr>
<td>size in bytes to size in pages</td>
<td>(round down) btop convert</td>
</tr>
<tr>
<td>gain access to a device crash dump</td>
<td></td>
</tr>
<tr>
<td>a driver's message freeing</td>
<td>routines dump</td>
</tr>
<tr>
<td>introduction to driver entry point</td>
<td>routine /structure that specifies</td>
</tr>
<tr>
<td>type</td>
<td>routines intro</td>
</tr>
<tr>
<td>request for controller that does structure uio</td>
<td>SAMESTR test if next queue is same</td>
</tr>
<tr>
<td>drivers bp_iosetup create</td>
<td>scatter/gather /break up an I/O</td>
</tr>
<tr>
<td>drivers mp_iosetup create</td>
<td>scatter/gather I/O request</td>
</tr>
<tr>
<td>noenable prevent a queue from being scheduled</td>
<td>scatter/gather list for block</td>
</tr>
<tr>
<td>space</td>
<td>scatter/gather list for STREAMS</td>
</tr>
<tr>
<td>segmap</td>
<td>scheduled</td>
</tr>
<tr>
<td>map device memory into user</td>
<td>segmap map device memory into user</td>
</tr>
<tr>
<td>putctl</td>
<td>send a control message to a queue</td>
</tr>
<tr>
<td>one-byte parameter to a/ putctll</td>
<td>send a control message with a</td>
</tr>
<tr>
<td>reverse direction qreply</td>
<td>send a message on a stream in the</td>
</tr>
<tr>
<td>putnext</td>
<td>send a message to the next queue</td>
</tr>
<tr>
<td>enableok reschedule a queue for service</td>
<td>service queued messages</td>
</tr>
<tr>
<td>srv</td>
<td>set the map's wait flag for a</td>
</tr>
</tbody>
</table>

DDI/DKI Reference Manual
allocate a message block using a
ptob convert size in pages to
(round down) btop convert
(round up) btopr convert
btop convert size in bytes to
btopr convert size in bytes to
ptob convert
size return

pending execution of an event
management map rmfree release free
bp_mapin allocate virtual address
deallocate virtual address
management map rmalloc allocate
kmem_alloc allocate
kmem_zalloc allocate and clear
allocate space from a private
free space back into a private
rminit initialize a private
segmap map device memory into user
drv_usecwait busy-wait for
timeout execute a function after a
delay delay process execution for a
bcanput test for flow control in
flushband flush messages for a
freeing/ free_rtn structure that
start

uio(D4DK) iovec data
qreply send a message on a
limit value structure module_info
create scatter/gather list for
structure streamtab
msgb
data
information structure qband
structure qinit
queue
declaration structure
driver
queue or band of the queue
queue or band of the queue
buf block I/O data transfer
cred access credential

shared buffer esballoc ............................................ esballoc(D3DK)
size in bytes .......................................................... ptob(D3DK)
size in bytes .......................................................... btop(D3DK)
size in bytes .......................................................... btopr(D3DK)
size in pages ......................................................... btop(D3DK)
size in pages ......................................................... btopr(D3DK)
size in pages ......................................................... ptob(D3DK)
size of logical device ............................................... size(D2D)
size return size of logical device ................................ size(D2D)
sleep suspend process activity .................................. sleep(D3DK)
space back into a private space ................................ rmfree(D3DK)
space ................................................................. bp_mapin(D3DK)
space ................................................................. bp_mapout(D3DK)
space from a private space ...................................... rmalloc(D3DK)
space from kernel free memory ................................ kmem_alloc(D3DK)
space management map ......................................... rmalloc(D3DK)
space management map ......................................... rmalloc(D3DK)
space management map ......................................... rinit(D3DK)
space ................................................................. segmap(D2K)
specified interval ................................................... drv_usecwait(D3DK)
specified length of time ........................................... timeout(D3DK)
specified number of clock ticks ................................ delay(D3DK)
specified priority band .......................................... bcanput(D3DK)
spl block/allow interrupts ..................................... free_rtn(D4DK)
srv service queued messages .................................. srv(D2D)
start access to a device .......................................... start(D2D)
start access to a device .......................................... start(D2D)
storage structure for I/O using ................................ iovc(D4DK)
strategy perform block I/O ..................................... strategy(D2D)
stream in the reverse direction ............................... qreply(D3DK)
STREAMS driver identification and ....................... module_info(D4DK)
STREAMS drivers mp_iostatup ............................... mp_iostatup(D3DK)
STREAMS entity declaration ................................. streamtab(D4DK)
STREAMS message block structure ......................... msgb(D4DK)
STREAMS message data structure .......................... datab(D4DK)
STREAMS queue flow control ................................. qband(D4DK)
STREAMS queue processing procedures ................... qinit(D4DK)
STREAMS queue structure ..................................... queue(D4DK)
streamtab STREAMS entity ................................. streamtab(D4DK)
strlog submit messages to the log ........................... strlog(D3DK)
strqget get information about a ............................ strqget(D3DK)
strqset change information about a ........................ strqset(D3DK)
buf block ............................................................ buf(D4DK)
cred access ........................................................ cred(D4DK)
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of the Association. To copy otherwise, or to republish, requires a fee and/or specific permission from the Association.

The Association of Computing Machinery, Inc.
1515 Massachusetts Avenue, NW
Washington, D.C. 20005-1209 USA

Copyright 79 ACM 0001-0010/79/0200...

ACM Reference Manual

DDI/DKI Reference Manual

Permuted Index

<table>
<thead>
<tr>
<th>datab STREAMS message data</th>
<th>datab(D4DK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iovec data storage</td>
<td>iovec(D4DK)</td>
</tr>
<tr>
<td>hdedata hard disk error data</td>
<td>hdedata(D4D)</td>
</tr>
<tr>
<td>map private memory map</td>
<td>map(D4DK)</td>
</tr>
<tr>
<td>identification and limit value</td>
<td>msgb(D4DK)</td>
</tr>
<tr>
<td>msgb STREAMS message block</td>
<td>msgb(D4DK)</td>
</tr>
<tr>
<td>convert page frame number to page queue flow control information</td>
<td>page_numtopp(D3DK)</td>
</tr>
<tr>
<td>STREAMS queue processing procedures</td>
<td>page_numtopp(D3DK)</td>
</tr>
<tr>
<td>STREAMS entity declaration</td>
<td>qband(D4DK)</td>
</tr>
<tr>
<td>message freeing routine free_rtn</td>
<td>qband(D4DK)</td>
</tr>
<tr>
<td>page_pptonum convert page</td>
<td>qinit(D4DK)</td>
</tr>
<tr>
<td>uio kernel data using uio(D4DK)</td>
<td>qinit(D4DK)</td>
</tr>
<tr>
<td>ureadc add character to a uio</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>remove a character from a uio</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>strlog</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>execution of an event sleep</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>completion of block I/O biowait</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>wakeup resume</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>dcache_sync an error message or panic the print display a driver message on priority band bcanput</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>SAMESTR test if next queue is same</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>message datamsg</td>
<td>queue(D4DK)</td>
</tr>
<tr>
<td>for a specified number of clock</td>
<td>testb(D3DK)</td>
</tr>
<tr>
<td>convert microseconds to clock</td>
<td>testb(D3DK)</td>
</tr>
<tr>
<td>drv_hztosec convert clock</td>
<td>testb(D3DK)</td>
</tr>
<tr>
<td>specified length of time</td>
<td>testb(D3DK)</td>
</tr>
<tr>
<td>utimeout cancel previous buf block I/O data</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>adjmsg</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>SAMESTR test if next queue is same</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>structure</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>ureadc add character to a uio</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>uwritec remove a character from a uio</td>
<td>timeout(D3DK)</td>
</tr>
<tr>
<td>uio scatter/gather I/O request</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio kernel data using uio(D4DK)</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) structure</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) iovec data</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) structure</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) iovec data</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) structure</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) iovec data</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) structure</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) iovec data</td>
<td>untimeout(D3DK)</td>
</tr>
<tr>
<td>uio(D4DK) structure</td>
<td>untimeout(D3DK)</td>
</tr>
</tbody>
</table>
The reference manual set for UNIX® System V Release 4 for Motorola Processors is the definitive source for complete and detailed specifications for all System V interfaces. Newly reorganized, this edition makes finding the manual page you need easy and fast. Each volume contains supplemental cross-references to aid those familiar with the old organization.

The new organization groups manual pages in the way most users need to use them:

- The Programmer's Reference Manual describes UNIX system calls and C language library functions, including new multiprocessing interfaces.
- The System Files and Devices Reference Manual describes file formats, special files (devices), and miscellaneous system facilities.
- The Device Driver Interface/Driver-Kernel Interface Reference Manual describes functions used by device driver software. Editions of this manual are available for both uniprocessor and multiprocessor versions of the operating system.
- The Master Index provides a master permuted index for the entire reference manual set.

Use Background Color To Locate Your Document Title:

<table>
<thead>
<tr>
<th>COLOR CODE</th>
<th>DOCUMENT TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GENERAL DOCUMENTS</td>
</tr>
<tr>
<td></td>
<td>USER'S GUIDES</td>
</tr>
<tr>
<td></td>
<td>ADMINISTRATOR'S GUIDES</td>
</tr>
<tr>
<td></td>
<td>PROGRAMMER'S GUIDES</td>
</tr>
<tr>
<td></td>
<td>REFERENCE MANUALS</td>
</tr>
</tbody>
</table>