第一章 引 论

1·1 关于本手册

《设备驱动程序界面/驱动程序-核心界面(DDI/DKI)参考手册》为开发人员在 Intel 80x86 体系结构下的 UNIX 系统 V 第 4 版 V 1·0 环境中编写设备驱动程序提供了参考信息。该手册介绍了两种设备驱动程序界面的规格说明:设备驱动程序界面(DDI)和驱动程序-核心界面(DKI)。按照这些界面中的一种或两者的要求所编写的驱动程序具有较强的可移植性。DDI 和 DKI 分别强调了与兼容性相关的不同方面(它们的区别请见图 1-1)。

图 1-1: DDI 和 DKI 概况

	由 SVR4 及以后版本支持	SVR4 支持
依赖于处理器的例程	DDI 专用 (DxD)	
独立于处理器的例程	DDI 和 DKI (DxDK)	DKI 专用 (DxK)

图 1-1 中的每一块表示了不同的界面集合。"DDI 专用"(在本手册中用 DxD 标记)依赖于特定的处理器,而且它能被 4·0 版之后的版本支持;本手册中所描述的这部分 DDI 是针对 INTEL 80X86 系列处理机,在 UNIX 系统 V 下专用的。"DKI 专用"(在本手册中用 DxK 标记)独立于处理器,但是在今后的版本中不一定支持。

本手册所描述的大部分例程、函数和结构同时属于"DDI 和 DKI"(在本手册中用 DxDK 标记)。如图 1-1 所示,同时满足 DKI 和 DDI 的驱动程序在所有支持 UNIX 系统 V 第 4 版 V1·0(Intel 80x86 体系结构)的 AT&T 计算机中是可移植的,而且它们均可运行于系统 V 第 4 版 V1·0 和以后的新版本。下面各部分将更详细地阐明 DDI 和 DKI 的可移植性和兼容性。

DDI 和 DKI 的作用是相互关联的,也就是说,一个驱动程序可以同时满足两种界面的要求,从而提高了驱动程序代码对操作系统新版本的可移植性和兼容性。

1.1.1 可移植性

如果一个软件经过较小的改动(其代价要比重写它低得多)即可运行于不同的环境中，那么该软件被认为是可移植的。该新环境可能包括不同的处理器、不同的操作系统、甚至不同的编程语言(如果存在一个语言翻译器)。不过，在大多数情况下，软件常常在相同的操作系统、处理器和源语言的环境中进行移植。在移植过程中，往往是修改源代码以使该软件能适应新的编译程序、新的处理器或者操作系统的新版本。

过去由于以下所述的原因，设备驱动程序很难进行移植，这些原因包括：

- 为增强功能，在由驱动程序访问的核心数据结构中增加了成员，或者重新定义了现有成员的大小。
- 核心函数的调用或返回语法已经改变。
- 驱动程序的开发人员不使用现有的核心函数，或者使用了未经正式规定的副作用，而新版本不再具有这样的副作用。
- 依赖于处理器的代码散布在驱动程序内的各处，而这种代码本应结合在一起构成独立的部分。

为了提高操作系统的性能，清除存在的问题以及增加新的特性，操作系统要定期更新版本；这或许是对兼容性的最大威胁。另一个问题是硬件的升级；随着硬件的更新换代，用户往往要采用更快、更高性能的同系列的计算机来取代现有的设备，尽管它们运行相同的操作系统，但是那些依赖于设备的代码也阻碍了软件的可移植性。

1.1.2 界面综述

虽然应用程序具有上述所有的与移植相关的问题，但是开发人员若要移植设备驱动程序则麻烦更大。在说明 DDI 和 DKI 的不同之处之前，我们必须了解设备驱动程序在 UNIX 系统中所处的地位。

设备驱动程序是控制数据传送到外部设备或从外设接收数据的核心模块。虽然驱动程序在 UNIX 系统中是作为核心的一部分进行配置的，然而它们的开发是独立于核心其它部分的。如果要使核心能被自由地修改，同时又保持与现有驱动程序的二进制兼容性，那么核心与驱动程序之间的交互式作用必须严格加以规范化。驱动程序/核心的服务界面是与驱动程序有关的三种界面中最为重要的一种。这三种界面为：

- 驱动程序—核心：I/O系统调用导致了对驱动程序入口点例程的调用，它们构成了服务界面的核心—驱动程序部分(如本手册第二章所述)。驱动程序可以调用第三章所描述的任意一个函数。这些函数是界面的驱动程序—核心部分。
- 驱动程序—硬件：所有的驱动程序(除了软件驱动程序)必须包括一个中断处理入口点，而且也可能进行直接存储区存取(DMA)。它们与其它依赖于硬件的交互式作用构成了驱动程序—硬件界面。
- 驱动程序—引导/配置软件：在系统引导时，由系统文件中的信息告诉系统存在着

某个驱动程序、使系统将该驱动程序包括进来、驱动程序与引导和配置软件间的交互作用是驱动程序涉及的第三个界面。参见《ISDG 参考手册》第三章中有关 ID/TP 的部分。

1·1·2·1 设备驱动程序界面(DDI)综述

DDI 的最基本的宗旨是:促进软件在一台特定机器上的 UNIX 系统 V 的所有后续版本之间实现源码和二进制码的可移植性。该宗旨隐含了一个非常重要的事实,尽管 DKI 只有一个,但是每个处理器都有属于它自己的 DDI。因此,若想将驱动程序移植到不同的硬件上,则需特别注意改动那些构成驱动程序的"DDI 专用"部分的与机器有关的例程。这些与机器有关的例程包括了如前面所述的驱动程序/硬件界面,但不只限于此;有些依赖于处理器的功能也可能属于驱动程序/核心界面。

为达到源码和二进制码的兼容性,DDI 中的函数、例程和结构的使用必须满足如下规则:

- 驱动程序不能直接存取系统状态结构(如u和sysinfo)。
- 对于那些驱动程序能直接存取的外部结构,只能使用本手册第三章所提供的那些(实用程序)函数。换句话讲,这些函数可以随处使用。
- 前导文件ddi.h必须包括在前导系统文件清单的末尾。该文件取消了一些宏定义、而用函数重新实现了与此相同的功能。在 ddi.h 之后应列出设备驱动程序包括的文件,从而确保驱动程序只使用 DDI/DKI 界面。

1·1·2·2 驱动程序-核心界面(DKI)综述

正如 DKI 的名字本身的含义那样,驱动程序-核心界面是为那些在核心和驱动程序之间进行通信的实用函数和入口点例程所定义的服务界面。它不包括驱动程序/硬件界面或驱动程序/引导软件界面。

驱动程序和核心之间的信息交换是以数据结构的形式进行的。DKI 规定了这些结构的内容,以及入口点和实用函数的调用与返回语法。

DKI 的目的是提高软件在不同计算机上的 UNIX 系统 V 环境下源码一级的可移植性。它仅仅适用于 Intel 80x86 体系结构下的系统 V 第 4 版 V1·0。由于 DKI 仅适用于驱动程序/核心的界面,所以,在移植过程中,与硬件和引导/配置界面有关的那部分驱动程序代码必须重写,而且应当尽可能构成独立的子程序。

注意: DKI中的某些界面不属于DDI。驱动程序开发人员应当清楚:不能保证系统V第4版之后的版本支持这些界面的使用。

1·1·3 界面成员

正如前面所述,本手册所描述的大部分入口点(第二章)、函数(第三章)和结构(第四章)同时属于 DDI 和 DKI。而下面的表 1-1 列出了那些仅属于 DDI 或仅属于 DKI 的入口

点、函数和结构。

表 1-1: 专用的入口点、函数和结构

	DDI 专用	DKI 专用
第二章	init,intr,halt,start	segmap,mmap
第三章	dma_pageio,dma_prog, dma_stop,dma_disable, dma_free_cb, dma_free_buf, dma_swsetup,dma_enable, dma_get_cb,dma_get_buf, dma_get_best_mode, dma_swstart,inb,outb, physiock,spl, repinsb,repinsw, repoutsd,inl,outl, repinsd,repoutsb, repoutsw,inw,outw	hat_getkpfnum
第四章	dma_buf,dma_cb	无

1.1.4 读 者

本手册的读者应当是那些在 AT&T UNIX 系统 V 第 4 版 V 1.0 (Intel 80x86 体系结构)及以后版本上建立、修改以及维护驱动程序的有经验的 C 程序设计人员。读者应当熟悉 UNIX 系统的核心和 C 程序设计语言的高级功能。本章"相关的学习材料"节中列出了可获得的 AT&T 文档及课程。

1.1.5 如何使用本手册

本手册分四章和两个附录:

- "第一章:引论"介绍了DDI、DKI以及其它的驱动程序界面,说明了本手册中所使用的各种符号约定以及与本手册相关的其它文档和课程。
- "第二章:驱动程序入口点"包含了所有驱动程序入口点例程的详细说明。
- "第三章:核心函数"包含了用于DDI/DKI驱动程序的所有驱动程序函数说明。
- "第四章:数据结构"包含了用于DDI/DKI驱动程序的数据结构的详细说明。
- "附录A:出错码"包含了所有与DDI/DKI驱动程序有关的错误代码。
- "附录B:从3.2版迁移到4.0版 V1.0"说明了UNIX 系统 V 3.2 版与UNIX 系统 V 4.0 版在 DDI/DKI 方面的不同之处。

1.2 驱动程序参考手册的组织

驱动程序参考手册中的"页"与《程序员参考手册》中的"页"类似，每个部分都由它所述的函数名和由括号括起的章序号开始，以便用户查找(如 canput (D3DK))。所有的驱动程序参考手册页条目都以"D"开头，表明这是驱动程序参考页。

目前，不同界面的手册页在出版时分在不同的卷中，每本手册都包含下面三部分：

D2　驱动程序入口点
D3　驱动程序所用的核心函数
D4　驱动程序存取的系统数据结构

各章序号后跟一个用于标记某种界面的后缀字符。这些后缀字符是：

D　设备/驱动程序界面(DDI)
K　驱动程序/核心界面(DKI)
DK　DDI 和 DKI

例如，open(D2DK)说明 open 是驱动程序的入口点例程，而不是《程序员参考手册》中的系统调用 open(2)。

1.3 本手册中所使用的约定

表 1-2 列出了本手册的一些印刷字体约定：

表 1-2: 本手册中所使用的印刷字体约定

项　目	字　型	例　子
C 语言保留字	定　宽	typedef
C 语言 typedef 说明	定　宽	caddr_t
驱动程序例程	定　宽	open 例程
出错值	定　宽	EINTR
文件名	定　宽	sys/conf.h
标志名	定　宽	B_WRITE
核心宏	定　宽	minor
核心函数	定　宽	ttopen
核心函数实参	斜体字	*bp*
结构成员	定　宽	b_addr
结构名	定　宽	buf 结构
符号常量	定　宽	NULL
系统调用	定　宽	ioctl(2)
C 函数库调用	定　宽	printf(3S)
Shell 命令	定　宽	layers(1)
用户定义的变量	斜体字	*prefix*close

1.4 相关的学习材料

为了支持用户使用我们的系统，AT&T 提供了大量的文档和培训课程。关于所提供材料的索引，可参见：

- *AT&T Computer Systems Documentation catalog* (300–000)
  (AT&T 计算机系统文档目录(300–000))
- *AT&T Computer Systems Education Catalog* (300–002)
  (AT&T 计算机系统培训目录(300–002))

1.4.1 文档

下面所涉及的大部分文档可从 AT&T 用户信息中心获得，订购时请使用括号内六位数的选择号。

若用电话订购，可使用下列电话号码：

1–800–432–6600　(美国本土且免费)
1–317–352–8557　(美国本土之外)

除了 AT&T 的文档外，下面还列出了一些可从其它商业渠道获得的相关文档。

注意：此处所列的某些文档是为UNIX系统V第3版而写的。由于这些文档对于理解驱动程序的操作环境和理论很有价值，因此建议用户在能够获得与 SVR4 相关的版本之前使用这些手册。若本手册的细节与下面所列文档的内容冲突，则以本手册为准。

1.4.1.1 驱动程序开发

《UNIX System V 和 V/386, R·3, 块和字符界面(BCI)开发指南》(307–191)，本书讨论了 UNIX System V R·3 的驱动程序开发的概念、性能、调试、安装及其它相关问题。

《UNIX System V 和 V/386, R·3, 块和字符界面(BCI)驱动程序参考手册》(307–192)，本书与上本手册同时使用，它包括了 UNIX System V R·3 参考材料。它叙述了驱动程序入口点例程(D2X)，用于 BCI 驱动程序的核心级函数(D3X)以及 BCI 驱动程序存取的数据结构(D4X)。

《UNIX System V PDI 驱动程序设计参考手册》(305–014)，本书定义了用于可移植的驱动程序界面(PDI)的驱动程序的核心函数和数据结构。

《UNIX System V SCSI 驱动程序界面(SDI)的驱动程序设计参考手册》(305–009)，本书定义了用于 SDI 驱动程序的核心函数和数据结构。

1.4.1.2 STREAMS

《程序员指南：STREAMS》讨论了如何为字符设备编写驱动程序，以及如何存取使用

STREAMS 驱动程序界面的字符设备。

1·4·1·3 C 编程语言和一般程序设计

Bentley, Jon Louis. *Writing Efficient Programs* (320-004), Englewood Cliffs, New Jersey: Prentice-Hall, 1982. 该书讨论了为提高处理性能的各种编程实践，它对于开发驱动程序很有帮助。

Kernighan, B. and D. Ritchie. *C Programming Language, Second Edition* (307-136), Englewood Cliffs, New Jersey: Prentice-Hall, 1988. 定义了 C 语言的函数、结构和界面，包括了一个简短的指导。

Lapin, J. E.. *Portable C and UNIX System Programming*, *Englewood* Cliffs, New Jersey: Prentice-Hall, 1987. 讨论了如何提高 C 语言程序的可移植性。

《程序员指南：网络界面》通过大量的实例为组成 UNIX 系统的传输层界面(TLI)的 3N 部分库函数提供了详细信息。

《程序员指南：ANSI C 和编程支持工具》讨论了如何使用 UNIX 公用程序(包括 make 和 SCCS)。

1·4·1·4 操作系统

Bach, Maurice J.. (320-044). Englewood Cliffs, New Jersey: Prentice-Hall, 1986. 讨论了 UNIX 操作系统的内部结构，包括解释如何驱动与核心函数相关的其它部分。

UNIX 系统 V 参考手册是 UNIX 操作系统的标准参考材料，它由三本独立的手册组成：

- 《系统管理员参考手册》包括系统的管理命令(1M)、特殊设备文件(7)和依赖于系统的维护命令(8)。
- 《程序员参考手册》包括程序设计命令(1)、系统调用(2)、库例程(3)、文件格式(4)和杂项信息(5)等。
- 《用户参考手册》包括UNIX系统的用户级命令(1)。

1·4·1·5 软件包装

《程序员指南：系统服务和应用软件打包工具》讨论了如何使用系统管理公用程序来编写安装驱动程序(或其它软件)所需的脚本。

1·4·2 培训

下面所列的培训课程是专门为驱动程序开发人员开设的。若有意参加，可拨如下电话号码：

- 在美国本土，拨1-800-TRAINER
- 在加拿大，拨1-800-221-1647
- 在美国本土之外，拨1-201-953-7554

"*C Language for Experienced Programmers* (UC 1001)"详尽正规地介绍 C 程序设计语言。

"*Internal UNIX System Calls and Libraries Using C Language* (UC 1011)"这是专门介绍在 UNIX 系统中进行 C 程序设计的课程，内容包括执行环境，存储区管理，输入/输出，记录锁和文件锁，进程生成以及进程间通信(IPC)等内容。

"*UNIX System V Release 4 Version 1·0 for the Intel 80x86 Architecture Device Drivers*· (UC 1056)"该课程主要讨论设备驱动程序机制，操作系统所提供的函数，设备驱动程序源码例子，安装过程和调试技术，内容包括字符设备、STREAMS 设备、块设备及整个 I/O 子系统。

"*UNIX System V Release 4 Version 1·0 for the Intel 80x86 Architecture Internals* (UC 1057)"该课程详细讨论 UNIX System V R·4·0 V· 1·0，包括进程子系统、文件子系统和 I/O 子系统，还包括 UNIX System V R·4·0 的新概念(如网络文件共享 NFS，快速文件系统以及虚拟文件系统)。

"*Internal System Calls and Libraries* (*Part 1*) (UC1058)"该课程讨论了 C 语言和 Intel 80x86 体系结构下的 UNIX System V R·4·0 V·1·0 的界面。该课程讨论了不与进程间通信相关的所有库函数和系统调用。和进程间通信相关的库函数和系统调用将在该课程的第二部分讨论。

"*Internal System Calls and Libraries* (*Part 2*) (UC1059)"它将讨论在 Intel 80x86 体系结构下的 UNIX System V R·4·0 V·1·0 中与进程间通信(IPC)相关的库函数和系统调用。

第二章　驱动程序入口点(D2)

2·1　引言

本章介绍了开发人员在编写设备驱动程序时所用到的入口点例程。它们可分为DDI/DKI共用、DDI专用和DKI专用三大类。这些例程之所以被称为入口点例程，是因为它们提供了从核心到驱动程序的调用语法和返回语法。对于所有类型的驱动程序而言，在执行系统调用时，在计算机启动时，或在设备产生中断时；或者对于STREAMS驱动程序，在处理STREAMS事件时，都要调用这些例程。

所有同时属于DDI和DKI的驱动程序例程用(D2DK)交叉引用代码加以标识，而所有的DDI专用或DKI专用的例程则用(D2D)或(D2K)交叉引用代码加以标识。

使驱动程序与核心进行通信的函数将在第三章说明。它们分别使用(D3DK)，(D3D)和(D3K)交叉引用代码标识。

在本章中，每个函数说明包括以下几部分：

- **名称**：说明例程的用途。
- **格式**：概述例程的调用和返回语法。
- **实参**：说明每个例程的实参。
- **说明**：说明该例程的一般情况。
- **返回值**：说明调用该例程后可能产生的返回值。
- **参见**：给出了解更多信息的相关材料。

2·2　驱动程序入口点例程和命名约定的说明

每个驱动程序由两部分组成：基础级和中断级。基础级用来与核心和用户程序进行交互作用；中断级用来和设备进行交互作用。

每个驱动程序由唯一的一个前缀串来标识，前缀串在驱动程序的主文件中定义。该驱动程序的每个例程名前都加上了这一前缀串。例如，硬盘驱动程序可以使用前缀 hd。因此，相应的例程应命名为 hdopen、hdclose、hdinit 等等。所有与该驱动程序相关的全局变量也都使用与此相同的前缀(更多的信息参见 ISDG 指南)。

系统例程能够调用由驱动程序开发人员命名的所有子例程。子例程应当被说明成静态的(static)，且应当使用驱动程序前缀以提高程序的可读性。

表 2-1 列出了本章将要说明的 STREAMS 驱动程序入口点，它们既可用于 DKI 又可

用于 DDI。

表 2–2 列出了本章将要说明的块设备 I/O 驱动程序入口点。除非加以特别说明，它们既可用于 DKI 又可以用于 DDI。

表 2–1: STREAMS 驱动程序入口点汇总

例　程	说　明
put	从前面的队列中接收消息
srv	对排队的消息进行服务

表 2–2: 非 STREAMS 专用的驱动程序入口点

例　程	说　明	类　型
chpoll	用于非 STREAMS 字符驱动程序的轮询入口点	
close	放弃对设备的存取权	
halt	当关闭系统时关闭设备	DDI 专用
init	初始化设备	DDI 专用
intr	处理设备中断	DDI 专用
ioctl	控制字符设备	
mmap	返回页框号	DKI 专用
open	获取对设备的存取权	
print	在系统控制台上显示驱动程序消息	
read	从设备上读数据	
segmap	将设备存储区映射到用户空间	DKI 专用
size	返回逻辑设备的大小	
start	启动对设备的存取	DDI 专用
strategy	执行块 I/O	
write	向设备写数据	

2.3 手册页

chpoll (D2DK)

名称

chpoll——非 STREAMS 字符驱动程序的轮询入口点

格式

```
#include <sys/poll.h>
int prefixchpoll (dev_t dev, short events, int anyyet, short *reventsp,
struct pollhead **php);
```

实参

dev 将被轮询的设备的设备号

events 可能发生的事件。有效的事件为:

POLLIN	数据可读
POLLOUT	可以不受阻塞地写数据
POLLPRI	高优先级数据可读
POLLHVP	设备挂起
POLLERR	设备出错

anyyet 标志位。若在pollfd数组中的任何其它文件描述字有事件挂起,则该标志位的值非零。poll(2)系统调用把指向一个 pollfd 结构型数组的指针作为它的一个实参。详细情况请参见 poll(2)手册页。

reventsp 指向满足条件的返回事件的位屏蔽的指针。

phpp 指向pollhead结构指针的指针。pollhead结构在sys/poll.h中定义。

说明

chpoll入口点例程由希望支持轮询的非STREAMS字符设备驱动程序使用,该驱动程序必须实施轮询规程。实现轮询规程时必须遵循下列规则:

1. 当chpoll被调用时,实现下列算法:

```
if (events_are_satisfied_now) {
    *reventsp = mask_of_satisfied_events;
} else {
    *reventsp = 0;
    if (! anyyet)
        *phpp = &my_local_pollhead_structure;
```

```
    }
    return (0);
```

2. 分配一个pollhead结构的实例。该实例可以与由驱动程序为每个次设备定义的数据结构相联系。驱动程序应当将 pollhead 结构看成"黑盒子"，该结构的任意一个字段都不能被引用。然而，在操作系统的各版本中该结构的大小须保持不变。
3. 只要发生上述任意一个events型的事件，就必须调用函数pollwakeup(D3DK)。该函数每次仅能由一个事件加以调用。

返回值

若成功，chpoll 例程返回零，否则，返回相应的出错号。

参见

pollwakeup (D3DK)，poll(2)。

close (D2DK)

名称

close——放弃对设备的存取权

格式 [块和字符]

```
#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);
```

实参

dev 设备号。

flag 文件状态标志，它由系统调用open(2)设置或由系统调用fcntl(2)加以修改。仅使用该标志信息——相应文件应当完全关闭。该标志从 file.h 中的 file 结构成员 f_flag 处获得，可能的值包括：FEXCL，FNDELAY，FREAD 和 FWRITE。详见 open(D2D)函数。

otyp 本参数使驱动程序能确定以何种原因打开了设备，以及打开了设备多少

次。以下标志假定 open 例程能被多次调用，但是 close 例程只能在最后一次关闭设备时调用。

OTYP_BLK	通过设备的块界面关闭
OTYP_CHAR	通过设备的原始/字符界面关闭
OTYP_MNT	调用close是系统调用umount(2)的结果。拆卸与块设备相关的文件系统。
OTYP_SWP	关闭对换设备
OTYP_LYR	关闭分级进程(高级驱动程序调用设备的close例程)

**cred_p* 指向用户凭证结构cred(D4D)的指针。

格式 [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);
```

实参

**q*	指向数据结构queue的指针，该结构用于引用驱动程序的读侧。(queue是一组结构和由其指向的一组例程的中心部分)。
flag	文件状态标志。
**cred p*	指向用户凭证结构cred(D4D)的指针。

说明

对于STREAMS驱动程序，close例程是由核心通过设备的cdevsw表项来调用的(模块则使用 fmodsw 表)。cdevsw 表项中 d_str 字段的非空值指向 streamtab 结构，而 streamtab 指向一个 qinit 结构；而 qinit 结构包含了指向 close 例程的指针。非 STREAMS 的 close 例程则直接通过 bdevsw(块)或 cdevsw(字符)表来调用。

close例程终止了用户进程和设备之间的连接，从而使软设备或硬设备做好被其它进程重新打开的准备。

一个设备能够同时被多个进程打开，而且每打开一次均要调用open驱动程序例程；然而核心仅仅在使用该设备的最后一个进程发出 close(2)或 umount(2)系统调用或

者退出时，才调用 close 例程。（例外的情况是：若 otyp 实参被置成 OTYP_LYR，那么在关闭时，对每一次打开都要调用一次 close（也有 otyp=OTYP_LYR））。

一般来说，close例程总是要检查*dev*参数的次设备号的合法性；必要时close例程也要检查存取权限（使用 cred(D4D)结构）以及 *flag* 和 *otyp* 参数值的合适性。

close 例程应当执行下列功能：

禁止中断
挂起电话线
磁带反绕
从私有缓冲方式中释放缓冲区
对不可共享的设备解锁（锁是在 open 例程中加上的）
清洗缓冲区
通知设备的关闭
释放在打开时分配的各种资源

在流被拆除以及模块被弹出时调用STREAMS驱动程序和模块的close例程。拆除流的步骤为：首先，任何一个和流相连接的多路转换器要被解除连结且最下层的流被关闭；其次，对于流中的每个驱动程序或模块执行下列工作（从流首到流尾）：

1. 给写队列一个机会，使其中的内容向后传送。
2. 调用close例程。
3. 从流中移去驱动程序或模块。

返回值

若成功，close例程应当返回零，否则，将返回相应的出错号（参见附录A所列的DDI/DKI出错码）。close 例程很少返回错误，但是，如果检测到错误，驱动程序则应当判断问题（或错误）的严重程度，并根据相应的错误程度，或在控制终端上显示出错消息，或（在最糟的情况下）产生系统的紧急情况。一般来说，close 例程发生错误的主要原因是与其相对应的设备出错。

参见

open(D2D)，cred (D4DK)。

halt (D2D)

名称

halt　当关闭系统时关闭驱动程序

格式

```
void prefixhalt ();
```

说明

在系统关闭时才调用 halt 例程。设备驱动程序不应当假定允许中断。

init (D2D)

名称

init — 设备的初始化

格式

```
void prefixinit ();
```

说明

init例程和start(D2D)例程用来对驱动程序及其所控制的设备进行初始化。init例程在系统初始化期间执行，且能在那些不需要低级系统服务来进行初始化的驱动程序中使用。start 例程仅当低级服务被启用后才能执行(比如中断和低级核心界面)，而且 start 例程应当在文件系统构造好之前执行。大部分的驱动程序既可使用 init 也可使用 start 例程，或者结合起来使用。

并不是所有的驱动程序都需要init例程或start例程。但是，当驱动程序需要在被打开之前分配任何数据结构时，就必须使用 init 例程或 start 例程。

init 例程和 start 例程可以执行下列任务，如：

为私有缓冲方式分配缓冲区
将设备映射到虚拟地址空间
硬件初始化(例如，系统生成和线路板复位)
字符驱动程序中串行设备的初始化

由于init和start是在形成用户环境之前运行的，所以不能调用需要用户环境的函数(如 sleep(D3DK))；而且，在启用中断之前执行 init 例程，因此 init 例程不能调用 spl (D3D)函数。

参见

start (D2D)

intr（D2D）

名称

intr——处理设备的中断

格式

void *prefix* intr(int *ivec*);

实参

ivec 操作系统为将驱动程序的中断处理程序同中断设备连接起来所使用的数字。*ivec* 的构成和解释依赖于每个系统的具体实现。在一些系统中，该数字可能是逻辑设备号，或者是逻辑设备号与逻辑控制器号的组合；它用于将正确的中断例程映射到相应的子设备上。在另外一些系统中，*ivec* 可能是中断向量号。

说明

intr例程是块设备硬件驱动程序和字符设备硬件驱动程序的中断处理程序。该中断处理程序负责确定中断的原因，处理中断，以及唤醒在该中断上睡眠的任意底层驱动程序进程。例如，当磁盘驱动器按照读要求已将信息传输至主机后，该磁盘驱动器的控制器将产生一个中断。然后，CPU 确认该中断并调用与控制器和磁盘驱动器相对应的中断处理程序。该中断处理例程首先处理中断，然后唤醒等待数据的驱动程序底层进程；接着由驱动程序的底层部分将数据传至用户。

一般来说，大部分中断例程应当完成下列工作：

保持中断事件的记录。

若驱动程序所控制的设备并未产生中断（仅用于支持共享中断的系统），则立即返回。

解释中断例程的实参*ivec*。

拒绝不能由设备控制器处理的设备的请求。

处理无原因中断（称为假中断）。

处理所有可能的设备错误。

唤醒在该中断请求上睡眠的所有进程。

还有许多与设备和驱动程序类型有关的工作要由intr执行。例如，下面几种驱动程序要求它们的 intr 具有不同的处理功能：

块驱动程序从队列中释放请求，唤醒睡眠在I/O请求上的进程。

终端驱动程序接收和发送字符。

打印机驱动程序确认字符已被发送。

此外，intr例程的功能依赖于具体的设备。因此，在使用intr例程时，你必须了解该设备产生中断的确切的芯片组、设备控制和状态寄存器的位模式以及数据如何在用户

和计算机之间进行传输；所有这些都因所访问设备的不同而相异。

对于不对每个子设备使用单独的中断向量的智能控制器，它的intr例程必须访问完成队列以便于确定中断是由哪个子设备所产生的。同时，该 intr 例程也必须更新状态信息、设置/清除标志位、设置/清除出错标志等相关参数，以便于 intr 例程完成对作业的处理。该例程还应当能处理由空完成队列所标识的假完成中断。当 intr 例程执行完之后，它应当移动指针至完成队列中的下一项。

如果驱动程序调用了biowait (D3DK)或sleep(D3DK)来等待操作的完成，那么 intr 例程必须调用 biodone(D3DK)或 wakeup(D3DK)来唤醒相应的进程。

intr 例程仅仅用于硬件驱动程序，不能用于软件驱动程序。

小心：intr 例程不能进行下列工作：

- 包含对核心函数sleep的调用
- 使用调用sleep的有关函数
- 将中断的优先级降低到低于进入中断处理例程时的级别。
- 调用任何需要用户环境的函数或例程(即，如果它访问或修改与现运行进程相关的信息)

注意：若uio (D4DK)结构中的成员uio_segflg被设置成UIO_USERSPACE(指明用户空间和核心空间的传送)，则中断例程不能使用 uiomove (D3DK)。

参见

biowait (D3DK)、sleep(D3DK)、biodone(D3DK)、wakeup (D3DK)。

ioctl (D2DK)

名称

ioctl——控制字符设备

格式

```
#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);
```