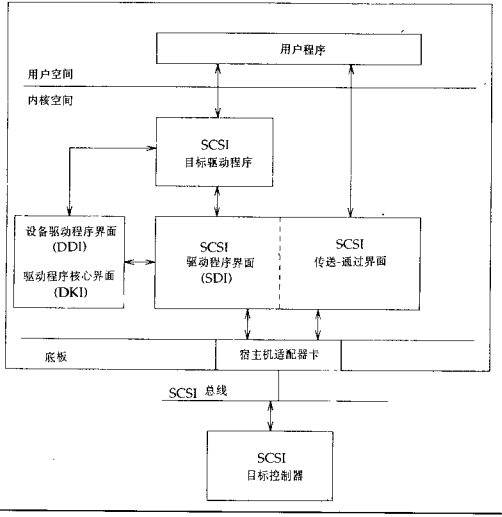
第一章 引 言

1·1 概述

1·1·1 SCSI 驱动程序界面

《程序员指南：SCSI 驱动程序界面(SCSI Driver Interface, SDI)》描述了一种与机器无关的机制，它用于编写访问 SCSI 设备的 SCSI 目标驱动程序(target driver)。图 1—1 说明了 SCSI 的硬件和软件。

图 1—1：SCSI 软件体系结构



SDI 与 SCSI 硬件互相配合，它为驱动程序或用户程序提供了一种访问 SCSI 设备的方法。

1·1·2 SDI 和设备驱动程序界面的关系

SCSI 目标驱动程序是用设备驱动程序界面(DDI)函数来编写的。DDI 是一个函数库。DDI 函数是作为 UNIX® 系统 V 版本 4 的一部分而包含在系统软件中。

SDI 增添了一些 DDI 没有的函数，这些函数提供了一组与 SCSI 设备交换命令和数据的服务及分配和释放数据结构的服务。SDI 包括了 I/O 控制，它们给于用户程序访问 SCSI 设备和 SDI 的直接命令。目标驱动程序能够使用这些控制对于 SCSI 设备进行用户测试、状态确定和诊断。SDI 函数包含在"SCSI 支撑软件包"中，它是与 386 计算机的 SCSI 宿主机适配器功能卡(SCSI host adapter feature card)一起收到的。

用 DDI 和 SDI 函数合理编写的目标驱动程序，能在装备有 SCSI 的任何 386 系列成员间直接移植。当保持了与 SDI 和 DDI 函数及结构的严格一致时，驱动程序的可移植性是存在的。DDI 具有各种各样的特点，这使得驱动程序开发者在编写驱动程序时能较快地编写代码，编写好的驱动程序可用于各种计算机，并且便于维护。

SCSI 由硬件和软件两者组成。SCSI 硬件包括一块称为宿主机适配器的专用功能卡，以及称为 SCSI 总线的连接 SCSI 设备的电缆。宿主机适配器在目标驱动程序和 SCSI 总线之间提供了一条通信管道线。宿主机适配器把一台 AT&T 的 386 计算机连接到与机器无关的 SCSI 总线上。

1·2 使用本手册

本手册的阅读对象是希望编写或维护 SCSI 目标驱动程序的驱动程序开发者。

本手册的各章概要如下：

第一章　引言

对本手册进行了介绍。也叙述了专门的程序设计考虑。

第二章　SDI 的 I/O 控制

对专门的 SDI 的 I/O 控制进行了说明。

第三章　函数

用来编写驱动程序代码的 SDI 函数。

第四章　结构

创建驱动程序时用到的数据结构和用I/O控制发送命令时要用的数据结构。

1·2·1 名字引用的约定

本手册和开放体系结构文献系列的其它图书中都普遍采用一种引用编号模式。对于驱动程序例程名、结构名和函数名，用后缀代码来说明每一名字的由来。此代码的格式为：

(D*nti*)

这里:

D=驱动程序界面指示符

n=名称类型: 2= 驱动程序例程

3= 函数

4= 结构

i=(任选的)界面类型:

P= DDI

I= SDI

X= 块和字符界面(若省略,则引用的是驱动程序例)

例如,`sdi_send`(D3I)是一个SDI函数,`ver_no`(D4P)是一个DDI结构。

1·2·2 字体的约定

表1–1给出了本手册使用的约定。

表1–1: 本书中使用的约定

项 目	铅字字体	例 子
书名	斜体	*Programmer's Guide*
C的位运算符(|、&、~)	全部大写字母	OR
C的`typedef`说明	等线体	`dma_tuple`
驱动程序前缀	斜体	*prefix*`close`(D1)
驱动程序例程	等线体	`strategy`(D1) routine
出错值	全部大写字母	EINTR
文件名	等线体	/usr/include/sys/ddi·h
标志名	全部大写字母	B_WRITE
函数	等线体	sdi_send(D21)
函数变元	斜体	*bp*
键盘上的键	Key	CTRL-d
结构成员	等线体	`b_flags`
结构名	等线体	`ver_no`(D21) structure
符号常量	全部大写字母	CE_CONT
UNIX系统的C命令	带有节引用的等线体	ioctl(2)

1·3 程序设计的考虑

本节给出了有关如何编写 SCSI 目标驱动程序代码的资料。说明了使用 SDI 编制驱动程序例程与使用 DDI 编制驱动程序例程之间的差别。

1·3·1 SDI 驱动程序例程的考虑

在初始化和启动时，核心程序经由 bdevsw 和 cdevsw 调用例程，例程也可被其它例程调用。为 SCSI 目标驱动程序编写的驱动程序例程，具有与任何其它块驱动程序例程一样的属性，仅有两处例外，其一是用户必须包括一个 ioctl(D2)例程，其二是不可包括由核心调用的中断例程。由于 SCSI 外设不是直接连接在底板上的，故常规的中断例程(intr)从不会被调用，因此它是无用的。

在《设备驱动程序界面/驱动程序——核心界面(DDI/DKI)参考手册》的第二章中，给出了驱动程序例程的完整说明。

表 1-2 汇总了可用来编制 SCSI 目标驱动程序代码的驱动程序例程。

表 1-2: 目标驱动程序例程一览

例程名称	说 明
*prefix*close (*device-number, flag, otype, cred-pointer*)	关闭一台设备
*prefix*init()	初始化一台设备
*prefix*ioctl (*device-number, command, arg, mode, cred-pionter, roal-point-er*)	I/O 控制
*prefix*open (*device-number, flag, otype, cred-pointer*)	打开一台设备
*prefix*print (*device-number, string*)	显示错误
*prefix*read (*device-number,uio-pointer, cred-pointer*)	读数据
*prefix*start()	启动设备访问
*prefix*strategy (*buf-pointer*)	块设备 I/O
*prefix*write (*device-number, uio-pointer, cred-pointer*)	写数据

1·3·1·1 中断

SCSI 目标驱动程序的中断例程与典型的硬件驱动程序的中断例程不同，因为它们控制的硬件不是直接连接在 6386 计算机的 I/O 总线上的。硬件驱动程序具有由操作系统调用的中断例程，这种硬件驱动程序采用非 SCSI 界面，与直接连接到 6386 计算机的 I/O 总线上的硬件相关。

SCSI 目标驱动程序中的中断例程由宿主机适配器软件调用，该软件以指向 sb(D4I) 结构的指针作为输入实参。宿主机适配器控制 6386 计算机 SCSI 总线上的硬件。

在发送作业之前，将一值赋于 scb(D4I)结构的 sc_int 成员。当目标驱动程序完成时，驱动程序用 sb(D4I)结构(具体使用 sc_int 成员)，把 SCSI 中断例程的地址传送到宿主机适配器。目标驱动程序可以为不同的操作定义不同的中断例程。

1-3-1-2 I/O 控制(ioctl)

SCSI 目标驱动程序必须为 ioctl 例程提供对 B_GETDEV 和 B_GETTYPE 这两条传送-通过界面(pass-through interface)命令的处理。下面的代码给出了一个例子，说明处理这两条命令的可能的办法:

```
#include "sys/types.h"
#include "sys/vtoc.h"
#include "sys/sdi.h"
#include "sys/scsi.h"
#include "sys/errno.h"
#include "sys/open.h"
#include "sys/sdi_edt.h"
dev_t ipt_dev;
    ...
switch (ioctl-cmd);
    ...
        case B_GETTYPE:
          if (cipyout)"scsi",
            ((struct bus_type *) arg)->bus_name, 5))
          {
            return (EFAULT);
          }
          if (copyout ("driver-prefix",
            ((struct bus_type *) arg) ->drv_name, 5))
          {
            return (EFAULT);
          }
          return (0);
        case B_GETDEV:
          sdi_getdev (&dk->dk_addr, &ipt_dev);
```

```
    •
if (copyout (&ipt_dev, arg, sizeof (ipt_dev)))
{
    return (EFAULT);
}
return (0);
```

bus_type 结构在/usr/include/sys/sdi_edt.h文件中定义。本例中，"5"表示"scsi"和一个结束符 NULL 共计 5 个字符，并且假定采用一个 4 字符的驱动程序前缀和一个结束符 NULL。

1·3·2 SCSI 总线复位

在软件或硬件故障的情况下，能够复位 SCSI 总线。(复位意味着：使用 SCSI 总线的所有作业被中断，并且在没有完成的情况下返回，此时，scb 结构中的 sc_comp_code 成员设置为 SDI_RESET)。在复位之后，SCSI 目标控制器可根据外围设备的情况，以至多 5 秒钟的时间复位其自身。在这段时间内，所有的 SCSI 作业都被阻塞，且不在 SCSI 总线上发送。一旦超过该时间段，则继续总线上的作业处理。装备有不同的多宿主机功能(在目前的 8368 上不支持)的计算机则增加一秒，以发送立即命令来复得设备控制(SCSI 的 RESERVE 命令)。目标驱动程序复得控制，使挂起的作业请求能够完成，而不受总线上其它宿主机的妨碍。假如驱动程序未顾及这一复得控制的机会，那么，另一宿主机上的目标驱动程序就可访问该设备并把它留作己用。

1·4 阅读建议

用户必须有一本《AT&T SCSI (*Small Computer System Interface*) *Definition*》(选择码 305-013)手册。当发送命令到 SCSI 设备时(使用该手册中说明的传送-通过界面，或 sdi_send 或 sdi_icmd 函数)，就需要此手册。

第二章　SDI 的 I/O 控制

2·1　引言

SDI 提供了一组 I/O 控制，可以在用户程序中使用它们。这些控制允许一个程序：

- 获取来自目标驱动程序或宿主机适配器的信息。
- 复位 SCSI 总线或目标控制器。
- 发送命令到 SCSI 设备。这种控制又称为传送—通过界面。

大多数控制是在 SDI 中实施的，但是，有些控制必须由 SCSI 目标驱动程序提供。

本章叙述使用 SDI I/O 控制的 `ioctl`(2)系统调用的有关信息。

在表 2–1 中，"用途"一栏说明了 SDI 的 I/O 控制完成什么任务。"控制"一栏是 `ioctl`(2)控制名。"源"一栏说明了从何处提供信息(例如，若源是 TD，则目标驱动程序必须在其 `ioctl`(D2)例程中提供信息。"头文件"一栏指出了定义 I/O 控制的头文件名。所有文件都在`/usr/include/sys`目录中。

表 2–1：SDI 的 I/O 控制

用　途	控　制	源	头文件
获取信息	B_GETDEV	TD+	sdi_det.h
	B_GETTYPE	HA	sdi_edt.h
	B_GETTYPE	TD	sdi_edt.h
	GETVER	HA	scsi.h
	HA_VER	HA	sdi.h
复位总线或目标控制器	SDI_BRESET	HA	sdi.h
	SDI_TRESET	HA	sdih
发送命令	SDI_SEND	HA	sdi.h

注意：缩写：

TD=　SCSI 目标驱动程序　　HA=　SCSI 宿主机适配器

注意：为了发送 SDI I/O 控制，程序必须拥有超级用户权限。

`ioctl`(2)命令的格式为：

`ioctl` (*file–descriptor,control–name, argument*)

这里：

file–descriptor　由对SCSI宿主机适配器设备文件的`open`(2)调用所产生。

control–name　SDI I/O控制的名称。

argument I/O控制需要的任选的变元。

表 2-2 表明了 ioctl 系统调用的第二个和第三个变元，以及第一个变元的起源。ioctl(2)命令的第一个参元总是一个文件描述字，该字由先前的 open(2)调用传送过来。open(2)调用的文件路径(第一个)变元规定了 ioctl 文件描述字的起源。

表 2-2: SDI ioctl 变元

第一变元 起源	第二变元 控制名	第三变元
GEN	HA_VER	指向结构 ver_no 的指针
GEN	SDI_BRESET	长整型 ddi_dev_t——SCSI 总线上的一台设备的次设备号
PT	SDI_SEND	指向结构 sb 的指针
PT	SDI_TRESET	无

表 2-2 中缩写的含义为:

BRESET : 总线复位。

GEN: 特定设备的设备号。此为具有Oxff次设备号的宿主机适配器的主设备号。

HA: 宿主机适配器。

PT: 特定设备的传送-通过设备号。

TRESET: 目标设备控制器复位。

VER: 版本号。

2.1.1 宿主机适配器次设备号

表 2-3 说明了宿主机适配器驱动程序 I/O 控制使用的次设备号格式:

表 2-3: 宿主机适配器次设备号

内容:	0	控制器			逻辑单元		
位:	7	6	5	4	3	2	1	0

使用这种控制器编址，而不使用 SCSI 控制块中提供的地址。若寻址到宿主机适配器，则传送-通过或控制器复位的请求被拒绝。该次设备号仅仅为宿主机适配器作 SDI_SEND 和 SDI_TRESET 控制时使用。

次设备号 Oxff 默认为宿主机适配器驱动程序，在作诸如获取信息等的一般控制时，

必须使用它。

2.2 获取信息

这些 I/O 控制提供各种信息源它们能通过 ioctl(2)命令来获得。这种信息对于启动其它 I/O 控制是有用的，且对于验证当前软件或硬件版本的可用性也是有用的。

此信息包括:

- 宿主机适配器或目标驱动程序的总线类型和驱动程序类型(使用B_GETTYPE命令)。
- 目标驱动程序设备类型(使用B_GETDEV命令)。
- 宿主机适配器版本号(使用GETVER命令)。
- ver_no结构的内容(使用HA_VER命令)。此结构中含有宿主机适配器版本号、计算机类型和 SCSI 软件的版本号。

存取信息的过程为:

- 对于宿主机适配器或目标驱动程序特别设备文件执行 open(2)系统调用，来获取一文件描述字。
- 使用ioctl(2)命令，发送命令到宿主机适配器或目标驱动程序。

错误

SDI 在 errno(通过目标驱动程序返回的错误取决于驱动程序的设计)中可能返回下列错误。若不能将信息拷贝到用户空间，则 B_GETTYPE 和 HA_VER 返回 EFAULT。

GETVER 不返回错误。

示例

下例说明了获取信息 I/O 控制的使用方法。

```
char *special_device;          /*Path name of target device */
int file_des;                  /*File descriptor */
dev_t pass_thru_device;        /*Pass through device number */

/*
 * Open the special device file so that we may query the target
 * driver for the pass-through major minor numbers of the device.
 */
if (file_des = open (special_device, O_RDONLY)) < 0
{
        fprintf (stderr, "Special device file open failed0);
```

```
}

/*
 *Perform the B_GETDEV command to obtain the pass-through
 *device number for this particular device.
 */
if (ioctl (file_des, B_GETDEV, &pass_thru_device) < 0)
{
fprintf (stderr, "Call to get pass-through device number failed0);
}
```

2.3 复位总线或目标控制器

SDI 的 I/O 控制提供复位 SCSI 总线的命令和复位目标控制器的命令。

执行这些命令的方法是：

1. 使用B_GETDEV命令来确定控制此设备的宿主机适配器的主和次设备号。
2. 打开设备特别文件，获取宿主机适配器的主和次设备号。
3. 使用mktemp(3C)，生成唯一的宿主机适配器节点名，然后，把此新名附加在此设备名的路径名的末尾。
4. 使用mknod(2)，创建一般访问宿主机适配器的节点，使得传送-通过界面能被使用。注意：对于 SDI_BRESET，当用户使用 makedev 来形成设备号(作为 mknod 函数的最末一个变元)时，使用 Oxff 代替次设备号。对于 SDI_TRESET，则使用传送-通过次设备号。
5. 为供传送-通过界面进行存取，打开宿主机适配器节点。
6. 复位总线或目标控制器。

错误

SDI_BRESET不在errno或SCB.sc_comp_code中返回任何值。SDI_TRESET在 errno 中返回 EBUSY(设备上的作业尚未完成，或者设备尚未激活)。此外，在 sb 正被发送到 SDI 期间，SCB.sc_comp_code 可被设置成 SDI_PROGRES(作业未完成)。

示例

```
#define HATEMP    "HAXXXXXX"

char *special_device;    /* Path name of target device */
```

```
char *pass_through[];    /* Path name of pass-through device */
int file_des;            /* File descriptor */
dev_t pass_thru_device;  /* Pass-through device number */
/*
 *Open the special device file so that we may query the target
 *driver for the pass-through major/minor numbers of the device.
 */
if ((file_des = open (special_device, O_RDONLY)) < 0
{
        fprintf (stderr, "Special device file open failed\n");
}

/*
 * Perform the B_GETDEV command to obtain the pass-through
 */ device number for this particular device.
 */
if (ioctl (file_des, B_GETDEV, &pass thru device) <0)
{
fprintf (stderr, "Call to get pass-through device number failed\n");
}

(void) close (file_des);

/*
 * Generate the pass-through special device file for
 * this device using mktemp(). To create the node in
 * the same directory as the target special device file,
 * the new pass-through name is appended on to the
 * path name of the target special device file.
 */

/* Copy the special device file name into the pass-through name */
(void) strcpy (pass_through, special_device);

/* Search to the last '/'. Append the newly created pass-through
 * device file onto the end. If no '/' was found, then use the
```

```
 * current directory.
 */
if ((ptr = strrchr (pass_through, '/')) !=NULL)
{
        (void) strcpy (++ptr, mktemp (HATEMP));
}
else
{
        (void) strcpy (pass_through, mktemp (HATEMP));
}
/*
 * Make the pass-though device node using the pass-through
 * major number of the target device and the general use
 * pass-through minor number (0xff).
 */
if (mknod (pass_through, (S_IFCHR | S_IREAD|S_IWRITE),
        makedev (pass_thru_device.maj, 0xff)) <0
{
        fprintf (stderr, "Unable to make pass-through node\n");
}

/*
 * Now that the pass-through node has been created, perform
 * an open it so that the command can be issued.
 */
if ((file_des = open (pass_through, O_RDWR)) < 0)
{
        fprintf (stderr, "Open of pass-through device failed\n");
}

/*
 * Issue the bus reset ioctl to the Host Adapter driver.
 * Pass it the pass-through minor number as an argument
 * so it can determine which bus to reset.
 */
if (ioctl (file_des, SDI_BRESET, pass_thru_device.min) <0 )
```

```
{
        fprintf (stderr, "SCSI bus reset failed\n");
}
```

2.4 发送一条 SCSI 命令(传送-通过界面)

SCSI 命令传送-通过界面使用户程序能够直接存取 SCSI 设备。由于允许用户程序以类似目标驱动程序的方式进行工作，于是使构造特定于设备的请求所需要的指令开销能够从目标驱动程序中删去。特定于设备的请求的一个实例是格式化软盘所需要的指令。使用传送-通过界面，用户程序能够针对不同卖主的驱动器选择不同的指令软件包。不但许多指令能从一个驱动程序中删除，而且能使一个驱动程序在范围广泛的不同驱动器上工作。此外，当设备改变时，无须频繁地更新驱动程序。

传送-通过界面给予用户一种方法，使他们在评估新外设和控制器时不必开发一个驱动程序，无需检查设备状态和去除了驱动程序的重复代码。

2.4.1 传送-通过的使用考虑

小心：当用户程序打开传送-通过界面的存取时，所有其它针对设备使用目标驱动程序的作业都被阻塞，直至传送-通过存取被关闭为止。这意味着，在传送-通过存取期间的任何 read(2)或 write(2)使用都失败。此外，使用传送-通过的进程不能使用请求进行目标驱动程序存取的系统调用。若一进程打开传送-通过界面，且随后执行存取设备的系统调用，则该系统调用失败。当系统调用产生 SDI_SEND 时，进程将挂起(某些 ioctl 命令不会产生 SDI_SEND)。若用户的传送—通过界面存取需要进行设备读和写，则用户必须在启动传送-通过之前，或用 SDI_SEND 发 CDB 命令的方式来做这些操作(本章随后解释)。

在编写存取传送-通过界面的程序时，须注意下列事项：

- 一个主-次设备号对与每个逻辑单元相关联。
- 必须设置特别设备文件的使用权限，使得只有属主才拥有读—写权限。
- 只有一个进程能使一个特别设备文件打开。在设备文件被打开时，从目标驱动程序至逻辑单元的各作业处理均被挂起。对于 6386 计算机，这与条件核实之后发生的挂起一样。
- 唯一能够使用的sb_type值是ISCB_TYPE(sb_type是sb结构的一个成员)。

注意：传送-通过界面要求目标驱动程序包括对B_GETDEV和B_GETTYPE这两条ioctl命令的存取。有关目标驱动程序这两条命令的实现，在本手册的第一章中讨论。

传送-通过界面为直接发送 SCSI 命令到 SCSI 设备提供了一条命令。在《AT&T SCSI(小计算机系统界面)定义》手册(选择码为 305-013)中说明的任何命令都能直接地发送到 SCSI 设备。这些命令提供的功能的例子是:

- 数据拷贝
- 设备格式化
- 方式选择
- 禁止/允许介质搬移
- 保留/释放单元
- 读/写数据

通过建立一个命令描述字块(Command Descriptor Block,由 scs 和 scm 定义,并且在《ANSI 小计算机系统接口(SCSI),X3T9.2/82-2,修订版 17B》手册中描述)发送各命令。进一步的信息,请参阅本手册第四章中关于 SCS 和 SCM 结构的讨论。

使用 SDI_SEND 命令的方法为:

1. 打开目标驱动程序。
2. 使用 B_GETDEV 来获取传送-通过宿主机适配器设备号。
3. 关闭对目标驱动程序的存取。
4. 使用 mknod(2)来创建一宿主机适配器节点。
5. 打开传送-通过界面。
6. 创建 sb 和 CDB。
7. 发送命令到 SCSI 设备。

错误

SDI_SEND 能够在 errno 中返回下列错误:

- EBUSY——作业对于设备排队(因此,不能够存取该设备)。
- EFAULT——向/从用户程序拷贝一个sb或一个scb的尝试失败。
- EINVAL——sb.sb_type没有被设置为ISCB_TYPE,或者SCB.sc_mode被设置为不支持的 SCB_LINK 值。
- ENOMEM——不能够为来自于被发送的CDB的数据或返回的数据分配存储器。

SDI_SEND 能够用 SCB.sc_comp_code 返回下列值:

- SDI_HAERR(宿主机适配器故障或奇偶性错误)。
- SDI_PROGRES(在正将sb发送到宿主机期间,作业没有完成)。

示例

```
#define HATEMP "HA× × × × × ×"
  char *special_device;     /* Path name of target device */
  char *pass_through[]      /* Path name of pass-through device */
```

```
int file_des;              /* File descriptor */
dev_t pass_thru_device;    /* Pass-through device number */
struct sb sb, *sb_ptr;     /* SCST block and pointer */
struct scs scs;            /* SCST command block */
char buffer [512]          /* Data buffer area */

/*
 * Open the special device file so that we may query the target
 * driver for the pass-through major minor numbers of the device.
 */
if ((file_des = open (special_device, O_RDONLY)) < 0)
{
       fprintf (stderr, "Specialdevice file open failed\n");
}

/*
 * Perform the B_GETDEV command to obtain the pass-through
 * device number for this particular device.
 */
if (ioctl (file_des, B_GETDEV, &pass_thru_device) < 0)
{
 fprintf (stderr, "Call to get pass-through device number failed\n");
}

(void) close (file_des);

/*
 * Generate the pass-through special device file for
 * this device using mktemp(). To create the node in
 * the same directory as the target special device file,
 * the new pass-throrgh name will be appended on to the
 * path name of the target special device file.
 */

/* Copy the special device file name into the pass-through name */
(void) strcpy (pass_through, special_device);
```

```
/* Search to the last '/'. Append the newly created pass-through
 * device file onto the end. If no '/' was found, then use the
 * current directory.
 */
if ((ptr = strrchr (pass_through, '/')) != NULL)
{
        (void) strcpy (++ptr, mktemp (HATEMP));
}
else
{
        (void) strcpy (pass_through, mktemp (HATEMP));
}
/*
 * Make the pass-through device node using the pass-through
 * major number of the target device and the general use
 * pass-through minor number (0xff).          */
if (mknod (pass_through, (S_IFCHR|S_IREAD|S_IWRITE),
          makedev (pass_thru_device.maj, 0xff)) <0)
{
        fprintf (stderr,"Unable to make pass-through node\n");
}

/*
 * Now that the pass-through node has been created, perform
 * an open it so that the command can be issued.
 */
if ((file_des = open (pass_through, O_RDWR)) < 0)
{
        fprintf(stderr,"Open of pass-through device failed\n");

}
/*
 * Set of the SCSI command descriptor block. In this
 * example end command will be a read of block 256 on
 * logical unit 0.
```

```
 */
scs.ss_op = SS_READ;
scs.ss_lun = 0;
scs.ss_addr = 256;
scs.ss_len = 1;
scs.ss_cont = 0;

/* Set up the SB */
sb_ptr = &sb;
sb_ptr -> sb_type = ISCB_TYPE;

/* Fill in the command address and size, the data transfer
 * address and the amount of data to be transferred, and
 * set the transfer mode to be a read from the device. */
sb_ptr->SCB.sc_cmdpt = SCS_AD(scs);
sb_ptr->SCB.sc_cmdsz = SCS_SZ (scs);
sb_ptr->SCB.sc_datapt = data_buffer;
sb_ptr->SCB.sc_datasz = 512;
sb_ptr->SCB.sc_mode = SCB_READ;
/* Set the timeout to 5 seconds */
sb_ptr->SCB.sc_time = 5000;

/*
 * Issue the SDI_SEND ioctl to send the command to
 * the Host Adapter driver. the third argument is the
 * pointer to the SB.
 */

if (ioctl (file_des, SDI_SEND, sb_ptr) < 0 )
{
        fprintf (stderr, "SCSI read command failed\n");
}
```